Training Deep Neural Networks for Stereo Vision

A dissertation presented

by

Jure Žbontar

to

The Faculty of Computer and Information Science

in partial fulfilment of the requirements for the degree of

Doctor of Philosophy

in the subject of

Computer and Information Science

Ljubljana, 2016

APPROVAL

I hereby declare that this submission is my own work and that it contains no material written by another person, except where due acknowledgement is made in the text.

— Jure Žbontar —

July 2016

The submission has been approved by

Yann LeCun, PhD

Silver Professor of Computer Science, Neural Science, and Electrical and Computer Engineering

ADVISOR AND EXAMINER

New York University

Janez Demšar, PhD

Associate Professor of Computer and Information Science

COADVISOR AND EXAMINER

Faculty of Computer and Information Science

Igor Kononeko, PhD

Full Professor of Computer and Information Science

EXAMINER

Faculty of Computer and Information Science

Igor Grabec, PhD

Professor Emeritus

EXAMINER

University of Ljubljana

PREVIOUS PUBLICATIONS

The research reported in this thesis was publicly presented at a conference workshop and was published in a peer reviewed conference and a peer reviewed journal.

[1] Jure Žbontar and Yann LeCun. Computing the Stereo Matching Cost with a Convolutional Neural Network. Oral presentation, *Reconstruction Meets Recognition Challenge Workshop* at the *European Conference on Computer Vision (ECCV)*, Zurich, September 2014.

[2] Jure Žbontar and Yann LeCun. Computing the Stereo Matching Cost with a Convolutional Neural Network. *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1592–1599, Boston, June 2015.

[3] Jure Žbontar and Yann LeCun. Stereo Matching by Training a Convolutional Neural Network to Compare Image Patches. *Journal of Machine Learning Research* 17(65): 1-32, April 2016.

I certify that I have obtained written permission from the copyright owner to include the aforementioned published material in my thesis. I certify that the above material describes work completed during my registration as a graduate student at the University of Ljubljana.

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*
Jure Žbontar
Učenje globokih nevronskih mrež za problem stereo vida

POVZETEK

V pričujoči doktorski disertaciji predstavimo metodo za izračun cene ujemanja za problem stereo vida. Stereo podatkovne množice, na primer KITTI in Middlebury, so v zadnjih nekaj letih postale dovolj velike, da se lahko problema lotimo z metodami, ki temeljijo na učenju. Naš pristop temelji na uporabi globoke konvolucijske nevronske mreže in algoritma za nadzorovano strojno učenje. Učno množico zgradimo iz javno dostopnih stereo podatkovnih množic. Učni primer sestoji iz para slikovnih zaplat in pripada enemu izmed dveh razredov: pozitivnemu, ko sta slikovni zaplati v korespondenci in negativnemu, ko nista.

Predstavljeni sta dve arhitekturi konvolucijskih nevronskih mrež za učenje podobnosti. Prva arhitektura je hitrejša od druge, vendar je izračunana globinska slika v povprečju manj natančna. V obeh primerih je vhod v nevronsko mrežo par slikovnih zaplat, izhod pa mera podobnosti med njima. Obe arhitekturi vsebujeta konvolucijski nevronski mreži, ki slikovni zaplati predstavita z vektorjem značilk. Podobnost med slikovnima zaplatama je izračunana na vektorju značilk, namesto na svetlostih posameznih slikovnih elementov. Prva arhitektura vektorja značilk primerja s kosinusno podobnostjo, medtem ko druga arhitektura vektorja primerja z naučeno večnivojsko nevronsko mrežo.

Razvito metodo primerjamo z uveljavljenimi metodami na treh podatkovnih množicah – KITTI 2012, KITTI 2015 in Middlebury – in ugotovimo, da je naša metoda najnatančnejša na vse treh podatkovnih množicah.

Ključne besede: stereo, cena ujemanja, učenje podobnosti, nadzorovano učenje, konvolucijska nevronska mreža

University *of Ljubljana*
Faculty *of Computer and Information Science*
Jure Žbontar
Training Deep Neural Networks for Stereo Vision

ABSTRACT

We present a method for extracting depth information from a rectified image pair. Our approach focuses on the first stage of many stereo algorithms: the matching cost computation. We approach the problem by learning a similarity measure on small image patches using a convolutional neural network. Training is carried out in a supervised manner by constructing a binary classification data set with examples of similar and dissimilar pairs of patches.

We examine two network architectures for learning a similarity measure on image patches. The first architecture is faster than the second, but produces disparity maps that are slightly less accurate. In both cases, the input to the network is a pair of small image patches and the output is a measure of similarity between them. Both architectures contain a trainable feature extractor that represents each image patch with a feature vector. The similarity between patches is measured on the feature vectors instead of the raw image intensity values. The fast architecture uses a fixed similarity measure to compare the two feature vectors, while the accurate architecture attempts to learn a good similarity measure on feature vectors.

The output of the convolutional neural network is used to initialize the stereo matching cost. A series of post-processing steps follow: cross-based cost aggregation, semiglobal matching, a left-right consistency check, subpixel enhancement, a median filter, and a bilateral filter.

We evaluate our method on the KITTI 2012, KITTI 2015, and Middlebury stereo data sets and show that it outperforms other approaches on all three data sets.

Key words: stereo, matching cost, similarity learning, supervised learning, convolutional neural networks

ACKNOWLEDGEMENTS

I would like to thank my thesis advisor, Yann LeCun, for getting me excited about stereo vision, for his invaluable comments, and for inviting me to New York University. This work would not be possible without Janez Demšar, the thesis co-advisor, and Blaž Zupan, the head of the Bioinformatics Laboratory. I would also like to thank the members of the Bioinformatics Laboratory, where I spent the first five years of my PhD program, and the members of the CILVR Laboratory, where I spent the last two. Finally, I am grateful for the support of my friends and family.

— Jure Žbontar, Ljubljana, July 2016.

CONTENTS

I

Introduction

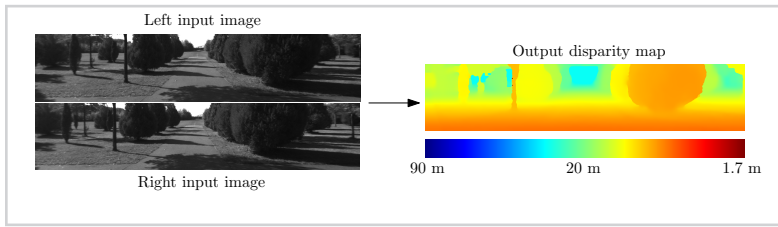


Figure 1.1

The input is a pair of images from the left and right camera. Note that objects closer to the camera have larger disparities than objects farther away. The output is a dense disparity map shown on the right, with warmer colors representing larger values of disparity (and smaller values of depth).

Consider the following problem: given two images taken by cameras at different horizontal positions, we wish to compute the disparity d for each pixel in the left image. Disparity refers to the difference in horizontal location of an object in the left and right image—an object at position (x, y) in the left image appears at position $(x - d, y)$ in the right image. If we know the disparity of an object we can compute its depth z using the following relation:

$$z = \frac{fB}{d}, \tag{1.1}$$

where f is the focal length of the camera, B is the distance between the camera centers. Figure 1.1 depicts the input to and the output from a dense two-frame stereo method.

When an object is photographed, information about its 3D structure, as well as its position, is lost. *Stereo vision* is the process of reconstructing the 3D model of the scene from two or more images by finding matching pixels in the images and converting their 2D positions into 3D depths. A stereo method attempts to invert the process of image formation—projecting points from the image plane back into the scene—to retrieve the original 3D structure of objects captured on film. Research on stereo methods dates back to 1970 and stereo continues to be an active area of research with several new algorithms published each year.

Learning based approaches are the main topic of this thesis. Two key factors—the introduction of large stereo data sets and the exponential growth of computing power—have enabled learning methods to overtake traditional stereo approaches in terms of accuracy. We focus on two-frame stereo methods and assume that the cameras differ in horizontal location, but are otherwise identical. While this assumption almost never holds for physical devices, it can be achieved by a post-processing step known as image rectification. We consider only dense stereo methods, which produce a depth

estimate at each pixel in an image.

The left and right images differ mostly in the horizontal location of objects; other differences are caused by reflections, occlusions, and perspective distortions. The horizontal displacement, or disparity, plays a crucial role in determining the 3D position of points in the scene, as objects closer to the image plane are displaced more than objects farther away. In fact, there is a direct relationship between disparity and depth, as we saw in Equation 1.1, and the problem of reconstructing a scene from two images reduces to the problem of determining disparities.

To estimate the disparity of a point in the left image a window-based stereo method crops a small patch around that point and searches for the most similar patch in the right image. The search for corresponding points needs to be carried out only in one dimension—along the epipolar line. There are several ways to define a similarity measure on image patches and the quality of the disparity map depends strongly on the similarity measure used. The main contribution of this thesis is a method that attempts to learn a similarity measure on image patches.

The 3D structure of a scene can, alternatively, be obtained using active sensors. A LIDAR sensor illuminates the target scene with laser light and calculates the distance by measuring the time required for the signal to return. Structured light is the process of projecting know patterns onto the scene and determining the disparity from multiple images. The Microsoft Kinect sensor projects an infrared speckle pattern, invisible to the human eye, onto the scene to determine its 3D structure using only a single camera.

Stereo methods have some advantages over active sensor: unlike LIDAR, stereo methods produce a dense disparity map; unlike the Kinect sensor and the structured light method, stereo methods can be used everywhere, indoor and outdoor; using a stereo method is usually less expensive and systems that require depth information usually have cameras already installed; the cameras themselves are typically smaller in size than some active sensors; and, since only cameras are used, multiple stereo systems do not interfere with each other. On the other hand, there are some advantages of using active sensors: they are typically more accurate than stereo methods and can be used in challenging conditions such as low light.

The output of a stereo algorithm is a dense disparity map, which can be used in many applications such as obstacle detection; object recognition and localization; autonomous driving; navigation; 3D mapping; image post-processing, for example, refocusing and background subtraction; intermediate view generation; and cartography.

1.1 Scientific Contributions

The main scientific contributions of this thesis are two new methods for computing the stereo matching cost. The contributions are itemized in the following list:

- *The accurate architecture—a convolutional neural network for computing the stereo matching cost optimized for accuracy.*

  The accurate architecture consists of two convolutional sub-network that extract feature vectors from small image patches. The feature vectors are forward-propagated through a multi-layer neural network that computes their similarity. The accurate architecture is trained on the KITTI and Middlebury stereo data sets to produce a good matching cost. Several post-processing steps are applied to improve the quality of the disparity maps. The error rates of the accurate architecture are 2.43 % on KITTI 2012, 3.89 % on KITTI 2015, and 8.29 % on the Middlebury data set. At the time of publication (November 2015), these were the lowest error rates on all three data sets.

  The proposed method was orally presented in the Reconstruction Meets Recognition Challenge workshop at the European Conference on Computer Vision 2014 in Zurich and was published in the Computer Vision and Pattern Recognition conference 2015 [154], which took place in Boston. An improved version of the accurate architecture is described in our Journal of Machine Learning Research paper in 2016 [155].

- *The fast architecture—a convolutional neural network for computing the stereo matching cost optimized for speed.*

  The accurate architecture produces precise disparity maps, however its range of application is limited to those that do not require real-time processing. After analyzing the network, we discovered that most of the time is spent in the multi-layer neural network with only a small fraction of time spent in the convolutional sub-networks. By replacing the multi-layer neural network with a fast operation—the dot product—we were able to compute the disparity maps up to 90 times faster. The error rates of the fast architecture are 2.82 % on KITTI 2012, 4.62 % on KITTI 2015, and 9.69 % on the Middlebury stereo data set. On small 320 × 240 resolution images with 32 disparity levels the output of

the fast architecture is computed in 30 milliseconds, which is 60 times faster compared with the 1.8 seconds required by the accurate architecture.

The fast architecture was first presented in our Journal of Machine Learning Research paper in 2016 [155].

- *The source code of both methods was made available under a permissive free software license.*

  The source code of both the fast and the accurate architecture is available online at https://github.com/jzbontar/mc-cnn under the permissive BSD licence. Several groups already use it in their work [1–3, 7, 54] and outperform our method on the KITTI and Middlebury data sets. At the time of writing (April 2016), the most accurate stereo methods on both the KITTI [54] and Middlebury [1] data sets use our code to compute the stereo matching cost. Releasing the source code was an important contribution to the field as it allowed other researchers to build on our work and push the boundaries of stereo matching.

The proposed stereo methods were thoroughly tested and compared with existing approaches. We evaluated different data augmentation steps, measured the runtime over many hyperparameter settings, compared the error rate against established matching cost functions, evaluated the effect of each step of the post-processing, simulated experiments on smaller data sets, measured the network's performance on the transfer learning setting by training on one data set and testing on a different data set, and compared many different hyperparameter settings.

2

Background

2.1 Neural Networks

A *neural network* is a function mapping input vectors to output vectors, parameterized by a set of weights. Given a training set of input and output pairs, the weights are adapted so that the outputs of the neural network match the outputs in the training set as closely as possible. A neural network consists of a network of processing units called neurons, hence the name.

Neural networks have been studied since the early 1940s [96, 108, 109, 146], at first as an attempt to model the biological processes in the brain. In this section, however, we examine neural networks from a statistical modelling point of view and do not concern ourselves with their biological plausibility.

LeCun et al. [90] and Schmidhuber [116] provide comprehensive surveys of the field. This section closely follows the treatment of neural networks given by Bishop [15].

2.1.1 Feedforward Neural Networks

A *feedforward neural network*, also known as a *multilayer perceptron*, is a neural network, where the processing units are partitioned into layers, with the outputs of one layer connected to the inputs of the next layer.

Two-layer Feedforward Network

We first compute M weighted sums of input variables $x_1, \ldots, x_D$:

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}, \tag{2.1}$$

where $w_{ji}^{(1)}$ and $w_{j0}^{(1)}$ are the weights and biases of the first layer. The M computed entities, a_j, are known as *activations*. The activations a_j are transformed with a nonlinear function $h(\cdot)$, known as the *activation function*:

$$z_j = h(a_j) \tag{2.2}$$

Many different activation functions are used in today's neural network architectures. Some of the more common activations functions are

- the logistic sigmoid activation function: $h(t) = \frac{1}{1+e^{-t}}$,

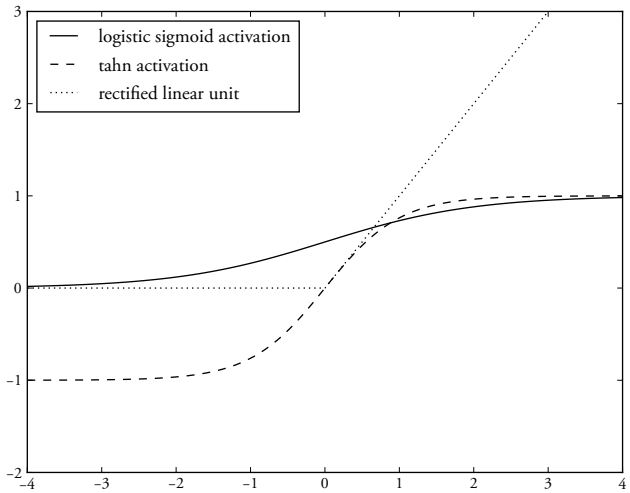


Figure 2.1

The logistic sigmoid activation, tanh activation, and rectified linear unite plotted on a graph.

- the tanh activation function: $h(t) = \frac{e^t - e^{-t}}{e^t + e^{-t}}$, and
- the rectified linear unit: $h(t) = max(0, t)$

The activation function are depicted in Figure 2.1. The rectified linear unit is the prevalent activation function in modern deep neural networks and plays a crucial role in achieving good performance [74].

The entities z_j are the outputs of the first layer. In the context of neural networks they are known as *hidden units*. The values of the hidden units are combined to give the K output unit activations:

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}, \tag{2.3}$$

where $w_{kj}^{(2)}$ and $w_{k0}^{(2)}$ are the weights and biases of the second layer.

An appropriate nonlinear function $g(\cdot)$ transforms the activations on the last layer to produce the final outputs y_k. The activation function at the last layer depends on the

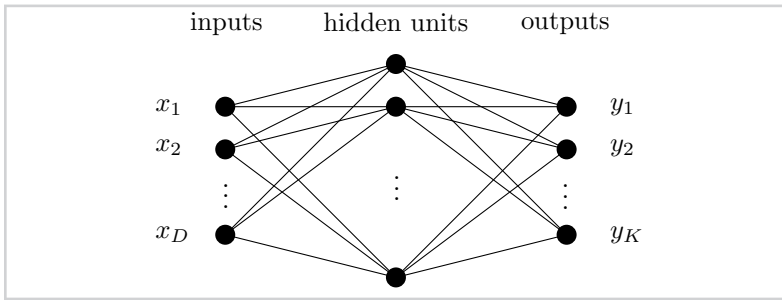


Figure 2.2

A two-layer neural network can be represented as a network diagram. The edges in the network correspond to weights and nodes to processing units.

nature and the distribution of the target variables. For regression problems—where the targets are real numbers and, given the inputs, follow a normal distribution—the identity function is used, that is $g(x) = x$. On binary classification problems—where the targets categorical and follow a Bernoulli distribution—the logistic sigmoid function is more appropriate, that is, $g(x) = \frac{1}{1+e^{-x}}$.

If we combine the individual stages of computation, we derive a single equation that defines the output of a two-layer feedforward network. Let $\mathbf{x} \in \mathbb{R}^D$ be a vector of inputs and $\mathbf{w}$ be a set of weights partitioned into matrices $\mathbf{w}^{(1)} \in \mathbb{R}^{M\times(D+1)}$ and $\mathbf{w}^{(2)} \in \mathbb{R}^{K\times(M+1)}$. The k-th output of the network is defined as

$$y_k(\mathbf{x}, \mathbf{w}) = g(\sum_{j=1}^{M} w_{kj}^{(2)} h(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}) + w_{k0}^{(2)}). \tag{2.4}$$

The architecture of a two-layer feedforward network is sometimes represented as a network diagram, such as the one shown in Figure 2.2.

Two-layer feedforward networks can approximate any continuous function on compact subsets of $\mathbb{R}^K$ to arbitrary precision (Cybenko [34], Hornik [68], Hornik et al. [69]). While the universal approximation theorems are reassuring, care should be taken not to overestimate their value in practice. The universal approximation network allocates several neurons to each region of the input space and learns the correct output for that region. With this construction, the number of hidden layers required to approximate an arbitrary function grows exponentially in the dimension of the input space.

If, instead, we choose to use a network with more hidden layers, it can result in a reduction of the number of hidden units required [8]. Even though the universal ap-

proximation theorems state that one hidden layer is enough to represent any function, it is often more efficient to use a deeper network.

Multi-layer Feedforward Network

A two-layer feedforward network can be generalized to contain any number of hidden layers. We can view a two-layer feedforward network, described in Equation 2.4, as a composition of two function, $f^{(1)}(\cdot)$ and $f^{(2)}(\cdot)$. Ignoring the dependence on the weights $\mathbf{w}$, the output of a two-layer neural network is defined as

$$y(\mathbf{x}) = (f^{(2)} \circ f^{(1)})(\mathbf{x}), \tag{2.5}$$

with $\circ$ denoting function composition and $f^{(0)}(\cdot)$, a function from $\mathbb{R}^D$ to $\mathbb{R}^K$, and $f^{(1)}(\cdot)$, a function from $\mathbb{R}^K$ to $\mathbb{R}^M$, defined as:

$$f_j^{(1)}(\mathbf{x}) = h(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}), \tag{2.6}$$

$$f_k^{(2)}(\mathbf{z}) = g(\sum_{j=1}^{K} w_{kj}^{(2)} z_j + w_{k0}^{(2)}). \tag{2.7}$$

The subscripts j and k in $f_j^{(1)}(\mathbf{x})$ and $f_k^{(2)}(\mathbf{z})$ denote the j-th and k-th output of the function, respectively. The function $f^{(1)}(\cdot)$ is the mapping from inputs to activations of the hidden units and $f^{(2)}(\cdot)$ is the mapping from hidden units to outputs.

The generalization of two-layer feedforward networks to multi-layer feedforward networks follows naturally from Equation 2.5 by composing N, rather than two, functions:

$$y(\mathbf{x}) = (f^{(N)} \circ \ldots \circ f^{(1)})(\mathbf{x}), \tag{2.8}$$

for appropriate definitions of the function $f^{(1)}, \ldots, f^{(N)}$.

More precisely, let N denote the number of layers and $D_1, \ldots, D_N$ denote the number of units in each layer (D_1 is the dimension of the input vector $\mathbf{x}$). The k-th output

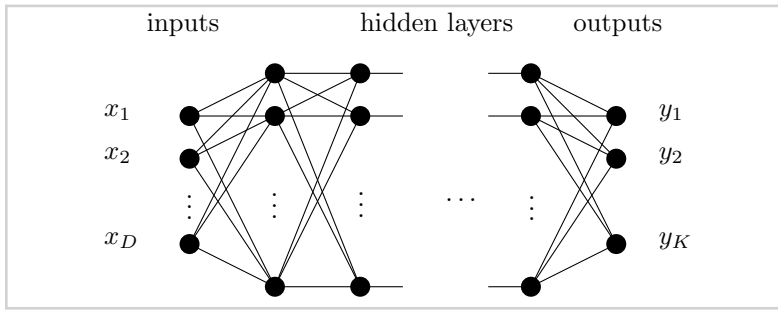


Figure 2.3

A multilayer-layer feed-forward represented as a network diagram.

y_k of a N-layer feedforward network is computed using the following equations:

$$z_j^{(0)} = x_j; \tag{2.9}$$

$$a_j^{(l)} = \sum_{i=1}^{D_l} w_{ji}^{(l)} z_j^{(l-1)} + w_{j0}^{(l)} \qquad \text{for } l = 1, \dots, D; \tag{2.10}$$

$$z_j^{(l)} = h(a_j^{(l)}) \qquad \text{for } l = 1, \dots, D-1; \tag{2.11}$$

$$y_k = g(a_j^{(N)}); \tag{2.12}$$

where $z_j^{(l)}$ denotes the j-th activation on the l-th layer and y_k denotes the k-tk output of the network.

If we define $z_0^{(l)} = 1$ for $l = 1, \dots, N$, we can simplify the notation by incorporating the bias term $w_{j0}^{(l)}$ into the sum. Equation 2.10 can be condensed to

$$a_j^{(l)} = \sum_{i=0}^{D_l} w_{ji}^{(l)} z_j^{(l-1)} \qquad \text{for } l = 1, \dots, D. \tag{2.13}$$

Note that the sum runs from $i = 0$ as opposed to $i = 1$ in Equation 2.10.

The function computed by a multi-layer feedforward network can be represented as a network diagram as shown in Figure 2.3.

2.1.2 *Network Training*

Network training refers to the process of determining a good set of weights $\mathbf{w}$. Intuitively, we would like the outputs of the network to match the desired outputs on the

training set. This notion is formalized by introducing the error function. The form of the error function depends on the type and conditional distribution of the target variable. In the following section we briefly overview *regression* problems, where the target is a continuous variable, and *binary classification* problems, where the target is a categorical variable.

Regression

Consider the case of a single target variable t that can take on any real value. We assume that the target variable t, given $\mathbf{x}$ and $\mathbf{w}$, follows a normal distribution:

$$p(t|\mathbf{x}, \mathbf{w}) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}) = \sqrt{\frac{\beta}{2\pi}} e^{-\frac{\{y(\mathbf{x},\mathbf{w})-t\}^2\beta}{2}}, \tag{2.14}$$

where $y(\mathbf{x}, \mathbf{w})$ is the output of the network with weights $\mathbf{w}$ on input vector $\mathbf{x}$ and β is the precision or inverse variance of the normal distribution.

Let $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ be a set of independent and identically distributed observations and let $\mathbf{t} = \{t_1, \dots, t_N\}$ be a set of corresponding target values, then the likelihood function is defined as

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^{N} p(t_n|\mathbf{x}^{(n)}, \mathbf{w}, \beta). \tag{2.15}$$

A standard way of determining the set of weights of a neural network is to select the vector $\mathbf{w}$ that maximizes the likelihood function. Maximizing the likelihood is equivalent to minimizing the negative log likelihood, because the logarithm is a monotonically increasing function. We prefer the negative log likelihood function because it simplifies the mathematical analysis, as well as helps prevent underflows when implementing the algorithm on a computer with limited precision floating point numbers.

From Equations 2.14 and 2.15, the negative log likelihood function can be written as:

$$\frac{\beta}{2} \sum_{n=1}^{N} \{y(\mathbf{x}^{(n)}, \mathbf{w}) - t_n\}^2 - \frac{N}{2} \ln \beta + \frac{N}{2} \ln(2\pi). \tag{2.16}$$

Leaving out the terms that don't depend on the weights $\mathbf{w}$, we arrive at the following definition of the error function $E(\mathbf{w})$:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \{y(\mathbf{x}^{(n)}, \mathbf{w}) - t_n\}^2, \tag{2.17}$$

which is referred to the *sum-of-squares* error function. The maximum likelihood solution is the vector of weights $\mathbf{w}$ that minimizes the value of the error function $E(\mathbf{w})$.

Because of the nonlinear activation function, the error function $E(\mathbf{w})$ is not convex and, as a result, obtaining the global minimum is not feasible in practice. Optimization algorithms will return a local minimum instead.

Binary Classification

In the case of binary classification the target variable t can take on one of two values: $t = 0$ for class C_- and $t = 1$ for class C_+. The activation function on the final layer of the network is chosen to be the logistic sigmoid function and the output of the network $y(\mathbf{x}, \mathbf{w})$ is guaranteed to lie on the interval $[0, 1]$. We interpret $y(t = 1|\mathbf{x})$ as the conditional probability of example $\mathbf{x}$ belonging to class C_+, that is $y(t = 1|\mathbf{x}) = p(C_+)$. The probability of example $\mathbf{x}$ belonging to class C_- is computed as $p(C_-|\mathbf{x}) = 1-p(C_+)$. The conditional distribution of the target variable t, given inputs $\mathbf{x}$ and weights $\mathbf{w}$, is given by a Bernoulli distribution:

$$p(t|\mathbf{x}, \mathbf{w}) = y(\mathbf{x}, \mathbf{w})^t\{1 - y(\mathbf{x}, \mathbf{w})\}^{1-t}. \tag{2.18}$$

The error function is obtained by taking the negative logarithm of the likelihood function. If the examples in the training set are independent and identically distributed, the error function takes the form:

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\{t^{(n)} \ln y^{(n)} + (1 - t^{(n)}) \ln(1 - y^{(n)})\}, \tag{2.19}$$

where $y(\mathbf{x}^{(n)}, \mathbf{w})$ is written as $y^{(n)}$ for brevity. This error function is referred to as the *binary cross-entropy* error function.

Parameter Optimization

Having defined the error function $E(\mathbf{w})$ for a regression task in Equation 2.17 and a binary classification task in Equation 2.19, we now consider the problem of searching for the weights $\mathbf{w}$ that minimize the error function $E(\mathbf{w})$.

Let $\nabla E(\mathbf{w})$ denote the gradient of function $E(\cdot)$ at $\mathbf{w}$. The gradient is the vector of partial derivatives of the error function with respect to each weight,

$$\nabla E(\mathbf{w}) = \left(\frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n}\right)^T. \tag{2.20}$$

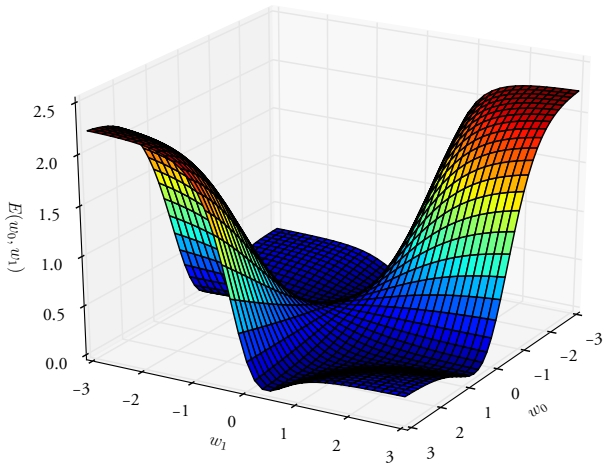


Figure 2.4

A graph plotting the error function $E(w_0, w_1)$ of a small network, with two weights.

The error function $E(\mathbf{w})$ can be viewed as a continuous surface sitting over weight space. Consider the extremely simple two-layer neural network with one input, one hidden layer, one output, no biases and the tanh activation function. The output function computed by such a network is defined as

$$y(x, w_0, w_1) = \tanh(w_1 \cdot \tanh(w_2 \cdot x)). \tag{2.21}$$

Furthermore, let the training set contain a single example with $x = 0.5$ and $t = 0.5$. The sum-of-squares error function $E(w_0, w_1)$ is defined as

$$E(w_0, w_1) = (0.5 - \tanh(w_1 \cdot \tanh(w_2 \cdot 0.5)))^2, \tag{2.22}$$

which can be plotted as a surface graph. See Figure 2.4.

Since the error function is not convex there will be, in general, many points where the gradient vanishes, $\nabla E(\mathbf{w}) = 0$. These points are called stationary and can be

classified into minima, maxima, and saddle points. This implies that finding the global minimum by obtaining an analytical expression for $\nabla E(\mathbf{w}) = 0$ is not possible, and we have to resort to iterative methods instead. Iterative methods are not guaranteed to find the global minimum of the error function $E(\mathbf{w})$. This might sound discouraging, however, experience and some recent theoretical results by Choromanska et al. [29] have shown that settling to a local minimum is good enough for successful application of neural networks.

Gradient Descent Optimization

One of the simplest and most widely used optimization methods for minimizing the error function of neural networks is gradient descent. Gradient descent is an iterative algorithm. The algorithm begins with an initial value of the weights $\mathbf{w}^{(0)}$ and performs a series of iterations, producing a new weight vector $\mathbf{w}^{(\tau)}$ at each iteration. In this section, we use the superscript notation $^{(\tau)}$ to denote iteration number. The initial weights $\mathbf{w}^{(0)}$ are typically chosen to be small random numbers. One strategy is to draw them from a zero-mean normal distribution with an appropriate standard deviation [84]. Several other initializations have been examined [51, 89, 110].

One iteration of gradient descent can be expressed with the following equation:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)}), \tag{2.23}$$

where $\eta > 0$ denotes the learning rate. At each iteration, the gradient of the error function is evaluated at $\mathbf{w}^{(\tau)}$—the current candidate for the weights. Since the gradient vector points in the direction of steepest ascent, we move in the opposite direction of the gradient, hence the minus sign in Equation 2.23. The learning rate η controls the amount of change applied to the weight vector at each iteration. It affects the number of iterations required before the method settles at a local minimum. A learning rate that is too small causes the method to converge slowly, while a learning rate that is too large causes the method to diverge.

The gradient computation $\nabla E(\mathbf{w}^{(\tau)})$ in Equation 2.23 involves iterating over the entire training set. Methods that have this property are referred to as batch methods. Alternative methods exist and have proven useful in practice, especially on large data sets. Recall from Equations 2.17 and 2.19 that the error function obtained by

maximum likelihood decomposes into a sum of terms, one for each training example,

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w}). \tag{2.24}$$

In stochastic gradient descent the update rule from Equation 2.23 is replaced by

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_n(\mathbf{w}^{(\tau)}). \tag{2.25}$$

At each iteration the weight update is based on the gradient of the error function computed on a single training example. Computing the gradient on a single example requires significantly less time than using the entire training set and, as a consequence, the weights are updated more frequently.

A compromise between batch and stochastic gradient descent is a method that computes the gradient on a subset of training examples. This is known as mini-batch gradient descent. The update rule for mini-batch gradient descent is

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \sum_{n \in B} \nabla E_n(\mathbf{w}^{(\tau)}), \tag{2.26}$$

where B denotes the set of examples in the mini-batch. One advantage of stochastic or mini-batch gradient descent over batch gradient descent is in the way they handle redundancy in the training set. Consider, as an extreme example, a new data set that is constructed by duplicating each training example. Note that the direction of the gradient is not affected; the new gradient is equal to the gradient on the original data set multiplied by 2. Batch gradient descent would require twice as much time to compute the gradient, whereas the two online methods would not be affected by the change. Another possible advantage of online methods is their ability to escape saddle points and local minima, because the error surface in online methods is constantly changing; at each iteration we follow the gradient of a different error function.

The optimization process can be improved by modelling inertia in the descent process. This is referred to as training with *momentum*. The weight update rule from Equation 2.23 is rewritten as the following two equations:

$$\mathbf{v}^{(\tau+1)} = m\mathbf{v}^{(\tau)} - \eta \nabla E_n(\mathbf{w}^{(\tau)}), \tag{2.27}$$

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \mathbf{v}^{(\tau+1)}, \tag{2.28}$$

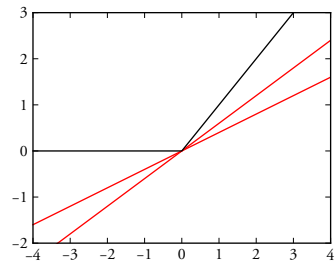


Figure 2.5

The output of a rectified linear unit in black plotted together with subtangent lines at $x = 0$ in red.

where we introduce a new vector of velocities $\mathbf{v}$. The vector $\mathbf{v}$ is the same size as the vector of weights $\mathbf{w}$ and contains the velocity of each weight in $\mathbf{w}$. The velocity vector is initialized to zero, that is, $\mathbf{v}^{(0)} = \mathbf{0}$. At each step, the velocities are attenuated by a factor $0 \leq m \leq 1$, known as the momentum term, and adjusted according to the gradient at that point. The weights are updated by adding the current candidate for the weights $\mathbf{w}^{(\tau)}$ and the velocity vector $\mathbf{v}^{(\tau+1)}$. Training with momentum helps in escaping local minima and increases the rate of change of weights whose partial derivatives don't switch sign often.

Subgradient Descent

A minor technical complication arises when applying gradient descent to a function that is not differentiable on all points in its domain. This happens, for example, when minimizing the error function of a neural network that contains rectified linear units or max pooling. In such cases, subgradient descent should be used in place of gradient descent. If the function is differentiable, subgradient descent and gradient descent return the same result. The subgradient descent method was developed by Shor [121].

A subderivative generalizes the derivative to functions that are not differentiable. A subderivative of a function $f : \mathbb{R} \to \mathbb{R}$ at point x_0 in the open interval I is a real number c such that

$$f(x) - f(x_0) \geq c(x - x_0) \tag{2.29}$$

for all $x \in I$. The intuition behind a subderivative is illustrated in Figure 2.5.

If f is convex, then the set of all subderivatives at x_0 is the non-empty closed interval $[a, b]$, where a and b are the left and right derivatives, respectively. For example, the set

of all subderivatives for a rectified linear unit $h(x) = \max(0, x)$ at $x = 0$ is the closed interval $[0, 1]$.

A subgradient is a generalization of a subderivative to functions with multiple inputs. A subgradient of a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ at $x_o \in \mathbb{R}^n$ in I is any vector $g \in \mathbb{R}^n$ such that

$$f(x) - f(x_0) \geq g^T(x - x_0) \tag{2.30}$$

for all $x \in I$.

The update step in subgradient descent is defined as

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta g(\mathbf{w}^{(\tau)}), \tag{2.31}$$

where $g(\mathbf{w}^{(\tau)})$ is any subgradient of the error function E at $\mathbf{w}^{(\tau)}$. Note the similarity with Equation 2.23, which defines the update step of gradient descent. The only difference is that the gradient $\nabla E(\mathbf{w}^{(\tau)})$ is replaced by the subgradient $g(\mathbf{w}^{(\tau)})$.

2.1.3 *Error Backpropagation*

Error backpropagation is an efficient method for computing the gradient of the error function with respect to the weights $\nabla E(\mathbf{w})$ (Werbos [144], Rumelhart [109], and LeCun [86, 87]).

We focus on computing $\nabla E_n(\mathbf{w})$, the gradient of the error function with respect to a single training example. The gradient with respect to the entire training set—or a batch of training examples—is obtained by summing $\nabla E_n(\mathbf{w})$ element-wise:

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N} \nabla E_n(\mathbf{w}) \tag{2.32}$$

As an introductory example, consider a feedforward network without hidden layers, and let the target $\mathbf{t}$ be a vector of real numbers. The output of the network y_k is computed as a linear combination of the inputs x_i:

$$y_k = \sum_i w_{ki} x_i, \tag{2.33}$$

where the bias term is included in the sum by defining $x_0 = 1$. The error function $E_n(\mathbf{w})$, derived from Equation 2.17, takes the form

$$E_n(\mathbf{w}) = \frac{1}{2} \sum_k (y_k^{(n)} - t_k^{(n)})^2, \tag{2.34}$$

where $y_k^{(n)} = y_k(\mathbf{x}^{(n)}, \mathbf{w})$ is the k-th output of the network on the n-th training example. The partial derivative of the error function E_n with respect to a particular weight w_{ij} is given by

$$\frac{\partial E_n}{\partial w_{ij}} = (y_j^{(n)} - t_j^{(n)})x_i^{(n)}. \tag{2.35}$$

A similar result can be obtained for the case of binary classification.

We now derive the equations for the partial derivatives in a multi-layer network. The output of each unit in a multi-layer feedforward network is given by Equations 2.13 and 2.11:

$$a_j = \sum_i w_{ji} z_i, \tag{2.36}$$

$$z_j = h(a_j). \tag{2.37}$$

We assume that a training example $\mathbf{x}^{(n)}$ has been forward-propagated through the network and that the quantities in Equations 2.36 and 2.37 have been computed and stored for each unit in the network.

The weight w_{ji} affects the error function E_n only through the value of a_j and we can use the chain rule to express $\frac{\partial E_n}{\partial w_{ji}}$ as a product of two terms:

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}}. \tag{2.38}$$

The second term $\frac{\partial a_j}{\partial w_{ji}}$ is equal to z_i and Equation 2.38 is rewritten as:

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i, \tag{2.39}$$

where we defined $\delta_j = \frac{\partial E_n}{\partial a_j}$. The δ terms are referred to as errors. Equation 2.39 implies that we can compute the gradient of the error function ∇E_n if we compute all of the error terms δ.

From Equation 2.35, the errors on the output layers of a feedforward neural network are computed as

$$\delta_k = y_k - t_k. \tag{2.40}$$

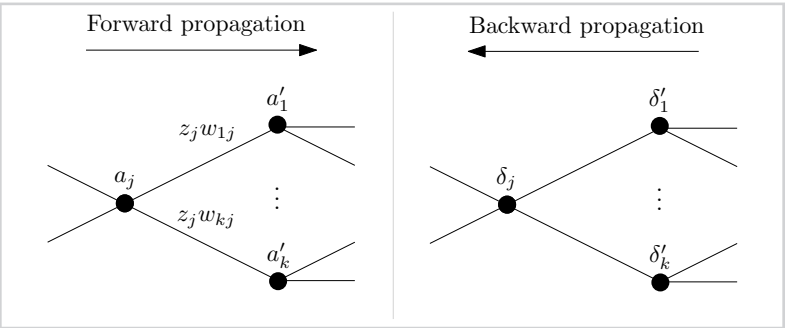


Figure 2.6

The values that are broadcast during forward and backward propagation.

This holds for both regression and binary classification problems, provided that the appropriate activation function is used at the last layer.

We turn our attention to computing the errors for the hidden units. Let $\delta_j = \frac{\partial E_n}{\partial a_j}$ denote the error of a hidden unit. The value a_j of a hidden unit is used to compute its activation z_j, which is consumed by units in the succeeding layer to compute a'_k. This is illustrated in Figure 2.6. The value a_j affects the error E_n only through the values of $a'_1, \ldots, a'_k$, and we can use the chain rule again to derive an equation for δ_j:

$$\delta_j = \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a'_k} \frac{\partial a'_k}{\partial a_j}. \tag{2.41}$$

The first term is the error of a unit in the next layer:

$$\frac{\partial E_n}{\partial a'_k} = \delta'_k. \tag{2.42}$$

The second term is derived by applying the chain rule yet again:

$$\frac{\partial a'_k}{\partial a_j} = \frac{\partial a'_k}{\partial z_j} \frac{\partial z_j}{\partial a_j} = w_{kj} h'(a_j). \tag{2.43}$$

Combining all the derived equations, we arrive at the final equation for the error term δ_j:

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta'_k. \tag{2.44}$$

Note that δ_j is computed only after all δ'_ks on the next layer are known.

To summarize, error backpropagation computes the gradient of the error function E_n with respect to the weights $\mathbf{w}$ for a single training example $\mathbf{x}^{(n)}$ using the following steps:

1. Forward propagate the input vector $\mathbf{x}^{(n)}$ using Equations 2.36 and 2.37.
2. Compute the errors δ_k for all output units using Equation 2.40.
3. Compute the errors δ_j for all hidden units using Equation 2.44.
4. Compute the partial derivatives with respect to the weights using 2.39.

The backpropagation algorithm can be applied to networks with a general structure, the only restriction being that there are no cycles in the network diagram, that is, the input of any processing unit must not depend on its output.

The Jacobian Matrix

The Jacobian matrix $\mathtt{J}$ of a function $f : \mathbb{R}^n \to \mathbb{R}^m$ is a $m \times n$ matrix of partial derivatives defined as

$$\mathtt{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}. \tag{2.45}$$

It is customary to break the implementation of a neural network system into modules. The output of the network is computed by composing the functions these modules implement. For example, a multi-layer neural network is composed of modules that compute the fully-connected layer from Equation 2.36, the activation functions from Equation 2.37, and a module that computes the cost function. Creating a new module consists of defining two function, one for the forward pass, called `fprop`, and one for the backward pass, called `bprop`[1]

The Jacobian matrix can be used to verify that the `bprop` function correctly computes the gradients. This is achieved by computing the Jacobian matrix twice, once by

[1]The backward pass is usually written as two separate functions, one for computing the gradient with respect to the inputs and one for computing the gradient with respect to the weights of the module. For modules that don't have weights (for example, the modules implementing activation functions) writing the former function suffices.

repeated calls to `fprop` and once by repeated calls to `bprop`. If the matrices obtained by the two approaches differ, that is, if the Frobenius norm of their difference is large, then we know there is an error in the implementation of either the `fprop` or the `bprop` function.

The Jacobian matrix can be computed from m calls to the `bprop` function as follows:

1. Choose an input vector $\mathbf{x} \in \mathbb{R}^n$ at which to compute the Jacobian matrix.
2. Use the `fprop` function to compute the output $f(x) \in \mathbb{R}^m$.
3. Repeat for $i = 1, \ldots, m$, once for each output:
   - Set the vector of incoming errors δ to zero, except for the i-th component, which you set to one.
   - Call `bprop` and store the results in the i-th row of the Jacobian matrix.

The Jacobian matrix can be computed from the `fprop` function with finite differences. This is achieved by perturbing each input and measuring how the perturbation affects the outputs:

$$\mathrm{J}_{ij} = \frac{\partial f_i}{\partial x_j} = \frac{f_i(x_j + \epsilon) - f_i(x_j)}{\epsilon} + O(\epsilon), \tag{2.46}$$

for a small $\epsilon > 0$. A small value of ϵ improves the accuracy of the approximation, while a large value of ϵ reduces numerical roundoff errors. $n+1$ calls to `fprop` are required to estimate the Jacobian matrix using Equation 2.46. The accuracy of the finite difference method can be improved by using symmetrical central differences which take the form

$$\mathrm{J}_{ij} = \frac{\partial f_i}{\partial x_j} = \frac{f_i(x_j + \epsilon) - f_i(x_j - \epsilon)}{2\epsilon} + O(\epsilon^2). \tag{2.47}$$

$2n$ calls to `fprop` are required to compute the Jacobian matrix using Equation 2.47.

2.1.4 *Convolutional Neural Networks*

Convolutional neural networks are a specific architecture of neural networks designed to process data that have a grid-like structure. Examples of input data with a grid structure include time-series or audio data, which can be thought of as a 1D grid;

images, which have a 2D structure; and video, which can be thought of as a series of images, and are therefore 3D. Convolutional neural networks should be used only if the input data satisfy the following two conditions.

First, neighbouring values of the grid should be correlated. In natural images, for example, this property holds, because neighbouring pixels are likely to have similar image intensity values. This implies that the number of image patches that occur in natural images is much smaller than the number of all possible image patches. This makes it possible to encode the information contained in an image patch by using fewer values than the number of pixels in the patch.

Second, local statistics of the input data should be the same on all positions on the grid. This property is best illustrated with an example. Consider natural images. Local motifs like edges, corners, and gratings can appear anywhere in an image. Feedforward neural networks do not take advantage of that and need to learn the appearance of a motif at each location independently.

Fukushima was the first to describe convolutional neural networks in his work on the *Neocognitron* [44–47]. The weights in the Neocognitron were set by local, winner-takes-all unsupervised learning rules or by pre-wiring. After training, the Neocognitron acquires a structure similar to the hierarchy model of the visual nervous system proposed by Hubel and Wiesel [71, 72]. LeCun [88] was the first to use gradient-based learning to set the weight of convolutional neural networks.

Convolutional neural networks have been used to solve diverse problems such as: image classification, localization, and detection [61, 74, 84, 119, 120, 122, 130]; face recognition [85, 133]; scene labeling [33, 40, 92, 159]; optical flow [42]; body tracking [135, 136]; image segmentation [30, 138]; signature verification [20]; obstacle avoidance [100]; learning image descriptors [153]; predicting surface normals [37]; monocular depth estimation [37, 38]; traffic sign recognition [31, 117]; and robotics [56, 118]. The section on convolutional neural networks is modeled after the textbook by Goodfellow et al. [52].

Convolutional Layer

The convolution of functions $x : \mathbb{R} \to \mathbb{R}$ and $w : \mathbb{R} \to \mathbb{R}$, denoted $x * w$, is defined as the integral of the product of the functions after one is reversed and shifted:

$$(x * w)(t) = \int_{-\infty}^{\infty} x(a)w(t-a)\,\mathrm{d}a. \tag{2.48}$$

In convolutional neural network terminology, the first term x is called the input, the second term w the kernel or weights, and the output is referred to as the feature map.

Functions $x(\cdot)$ and $w(\cdot)$ are usually—at least in a machine learning setting—discretized and provided as vectors of real values. We define the discrete convolution of vectors $\mathbf{x}$ and $\mathbf{w}$ as

$$(\mathbf{x} * \mathbf{w})(t) = \sum_a x_a w_{t-a} \tag{2.49}$$

We often use convolutions over more than one axis at a time, for example, when the input to the network is an image. Let I denote a matrix representing the input image and let K denote the matrix of weights or the kernel. The discrete convolution of matrices I and K is defined as:

$$(\mathbf{I} * \mathbf{K})(i,j) = \sum_m \sum_n I[m,n]K[i-m, j-n], \tag{2.50}$$

where we use $I[m,n]$ to denote the element in the m-th row and n-th column of matrix I. Because convolution is commutative we may write

$$(\mathbf{I} * \mathbf{K})(i,j) = (\mathbf{K} * \mathbf{I})(i,j) = \sum_m \sum_n K[m,n]I[i-m, j-n], \tag{2.51}$$

which is usually the way that convolution is implemented on a computer. Furthermore, many convolution libraries implement the cross-correlation function, which is the same as convolution without flipping the kernel:

$$(\mathbf{K} * \mathbf{I})(i,j) = \sum_m \sum_n K[m,n]I[i+m, j+n]. \tag{2.52}$$

From now on we use the term convolution to denote the discrete cross-correlation operation defined in Equation 2.52.

We focus only on the case of 2D input data, because this was the setting of our thesis problem. The input to and the output from a convolutional layer, as well as its weights, are stored in multi-dimensional arrays known as tensors.

Let z and a denote three-dimensional tensors of inputs and outputs. The first dimension indexes the feature map while the second and third dimensions index the spatial location[2]. Let w denote a four-dimensional tensor of weights. The first and

[2]Software implementations of neural networks typically represent z and a as four-dimensional tensors instead. The additional dimension is used to group together examples in mini-batches.

second dimensions index the weight—a different set of weights is used for each input-output feature map—while the third and forth dimensions the index spatial location. We use b to denote a one-dimensional tensor of biases. Tensors w and b constitute the trainable parameters of a convolutional layer.

The output tensor a of a convolutional layer given the input tensor z, weights w, and biases b is defined as

$$a[j,k,l] = b[j] + \sum_i (w[i,j,:,:] * z[i,:,:])(k,l) \tag{2.53}$$

$$= b[j] + \sum_i \sum_m \sum_n w[i,j,m,n]\, z[i,k+m,l+n], \tag{2.54}$$

where $w[i,j,:,:]$ denotes a two-dimensional tensor of the (i,j)-th weight and $z[i,:,:]$ denotes a two-dimensional tensor of the i-th input feature map. The output feature map j at location (k,l) is obtained by convolving each input feature map with the corresponding kernel at location (k,l), summing the results, and adding the bias.

A convolutional layer differs from a linear layer in two ways: (1) some weights in a convolutional layer are shared, which means that the same kernel is applied at all spatial locations of the input; (2) a neuron is connected only to a local neighbourhood of neurons from the preceding layer.

A convolutional layer is equivarient to translation. A function $f(\cdot)$ is equivarient to function $g(\cdot)$ if $f(g(x)) = g(f(x))$, that is, if the input to a function changes, the output changes in the same way. In the context of convolutional neural networks, $g(\cdot)$ represents translation and $f(\cdot)$ represents the convolutional layer defined in Equation 2.54. For example, let $g(z[i,k,l]) = z[i,k,l-1]$ represent a translation by one pixel to the right. If we shift the input and then apply convolution the results are the same as if we first apply convolution and then shift the output. A convolutional layer is not equivarient to other transformations such as rotation or scaling.

In a typical convolutional neural network, the convolution layer defined in Equation 2.54 is followed by a nonlinear activation function, typically the rectifying linear unit. Several convolutional layers may be chained together. A pooling and subsampling layer, described in the next section, is also often used. Figure 2.7 depicts a small segment of a larger convolutional neural network.

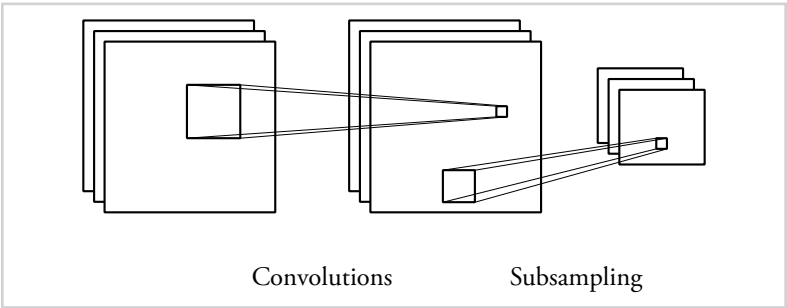


Figure 2.7

A small segment of a convolutional neural network, showing a convolutional layer followed by a sub-sampling layer.

Pooling and Sub-sampling Layer

A pooling layer computes summary statistics at nearby locations of the input tensor. Pooling may be used on data with a grid-like structure. Since we use pooling only on images in this work, we will focus on two-dimensional input data.

Let z and a denote three-dimensional tensors of inputs and outputs, with the first dimension indexing the feature map and the last two dimensions indexing the spatial location. A max pooling layer with a window size of d is defined as

$$a[j,k,l] = \max\{z[j,k+m,l+n] : m,n \in \{0,\dots,d-1\}\}. \tag{2.55}$$

The output of a max pooling layer of the j-th feature map at location (k,l) is defined as the largest value of the j-th input feature map in a $d \times d$ window with its upper left corner at (k,l).

It is possible to replace the max function in Equation 2.55 with other summary statistics, such as the average, an L2 norm, or a weighted average based on the distance from the central pixel.

The intuition behind the pooling layer is to introduce invariance into the model, so that small changes in the input don't change the output. Invariance is a useful property of the model; we often care more about whether or not some feature appears in the image than the exact location of the feature.

After the max pooling operation described in Equation 2.55 many neighbouring units will have the same value, due to the nature of the max function. To remove this redundancy, a sub-sampling layer usually follows. The output of the sub-sampling

layer with stride s is defined as

$$a[j,k,l] = z[j,sk,sl], \tag{2.56}$$

where z and a are three-dimensional tensors of inputs and outputs. The sub-sampling layer keeps every s-th value and discards the rest. The width and height of the output tensor a are s-times smaller than the width and height of the input tensor z, and, because of this reduction in spatial resolution, sub-sampling also improves the computational efficiency. While the sub-sampling stride s can be chosen to be the same as the max pooling window size d it is also common to use $s < d$ and have the pooling regions overlap. We have defined the max pooling and sub-sampling layers in two separate equations. Most implementations, however, combine these two operations to avoid computing max pooling outputs that will be discarded in the subsequent sub-sampling step.

2.2 Two-View Geometry

This section reviews the geometry of two perspective views and describes the process of image formation, epipolar geometry and derives the fundamental matrix for a parallel camera stereo rig. We follow the presentation given in the textbook *Multiple View Geometry in Computer Vision* by Hartley and Zisserman [60].

2.2.1 Camera Models

The process of image formation is modelled by a central projection. We use bold lowercase letters ($\mathbf{x}, \mathbf{y}, \mathbf{z}$) to denote 2D column vectors. Bold uppercase letters ($\mathbf{X}, \mathbf{Y}, \mathbf{Z}$) denote 3D column vectors. When convenient, we write the elements of the vector explicitly, such as (x, y, z), in which case, the vector represents a row vector; that is $\mathbf{X} = (x, y, z)^T$, where $(x, y, z)^T$ is the transpose of (x, y, z).

Homogeneous Representation of Lines A line in the plane is defined as the set of points (x, y) for which the following equation holds: $ax+by+c = 0$, where a, b, and c are real numbers. The same line can be represented as a 3D column vector $(a, b, c)^T$, but the correspondence between lines and vectors is not one-to-one. Consider, for example, vectors $k(a, b, c)^T = (ka, kb, kc)^T$ and $(a, b, c)^T$. The vectors represent the same line because $ax + by + c = 0$ if and only if $(ka)x + (kb)y + kc = 0$ for $k \neq 0$ and, in that

sense, the vectors $k(a,b,c)^T$ and $(a,b,c)^T$ are equivalent. Vectors with this equivalence relation are called *homogeneous* vectors.

Homogeneous Representation of Points A point in 2D Euclidean space is represented as an ordered pair of real numbers: $\mathbf{x} = (x,y)^T$. Let $\mathbf{l} = (a,b,c)^T$ represent a line in the plane. The point $\mathbf{x}$ lies on line $\mathbf{l}$ if and only if $ax + by + c = 0$. We can write this condition more succinctly using a single dot product $(x,y,1)\mathbf{l} = 0$, where the point $(x,y)^T$ is written as a 3-vector $(x,y,1)^T$ by appending a final coordinate 1. The new representation has an interesting property, namely that the point $(x,y,1)^T$ lies on line $\mathbf{l}$ if and only if the point $(kx,ky,k)^T = k(x,y,1)^T$ lies on line $\mathbf{l}$. It is, therefore, natural to associate $(x,y,1)^T$ and $(kx,ky,k)^T$ with the same point (x,y) in $\mathbb{R}^2$. Similar to the case with lines, points in 2D will often be represented as homogeneous 3-vector. A point $\mathbf{x} = (x_1,x_2,x_3)^T$ in homogeneous coordinates lies on a line $\mathbf{l} = (a,b,c)^T$ if and only if the following holds:

$$\mathbf{x}^T\mathbf{l} = 0. \tag{2.57}$$

To summarize, a point in $\mathbb{R}^2$ can be represented either as an *inhomogeneous* 2-vector (x,y) or a *homogeneous* 3-vector (kx,ky,k), for any non-zero value of k.

Line joining points Let $\mathbf{x}$ and $\mathbf{y}$ denote two points in homogeneous coordinates. The line $\mathbf{l}$ that passes through points $\mathbf{x}$ and $\mathbf{y}$ is defined as the cross-product

$$\mathbf{l} = \mathbf{x} \times \mathbf{y}. \tag{2.58}$$

To see this, recall some properties of the scalar triple product $\mathbf{a}^T(\mathbf{b}\times\mathbf{c})$, namely $\mathbf{a}^T(\mathbf{a}\times\mathbf{b}) = 0$ and $\mathbf{b}^T(\mathbf{a}\times\mathbf{b}) = 0$. Line $\mathbf{l}$ passes through point $\mathbf{x}$ if and only if $\mathbf{x}^T\mathbf{l} = 0$, which holds because $\mathbf{x}^T\mathbf{l} = \mathbf{x}^T(\mathbf{x}\times\mathbf{y}) = 0$. We can prove that line $\mathbf{l}$ passes through point $\mathbf{y}$ using a similar argument.

Cross product as a matrix-vector multiplication Let both $\mathbf{x} = (x_1,x_2,x_3)^T$ and $\mathbf{y}$ be 3-vectors. We wish to represent the cross-product $\mathbf{x}\times\mathbf{y}$ as a matrix-vector multiplication. We can achieve this by defining an operator that maps a 3-vector into a 3×3 matrix:

$$[\mathbf{x}]_\times = \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix}, \tag{2.59}$$

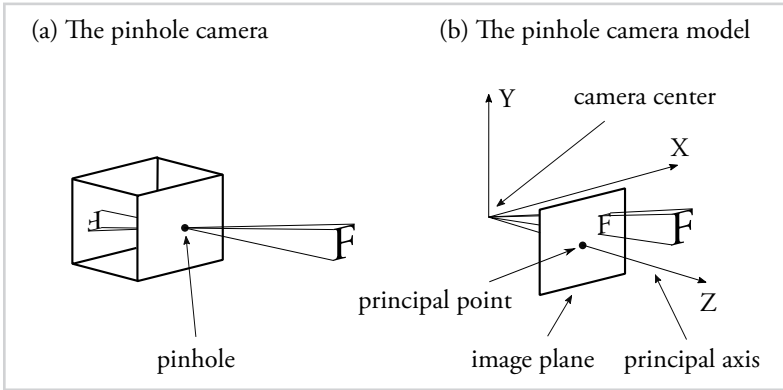


Figure 2.8

The pinhole camera and the pinhole camera model. In the camera model the image plane is in front of the camera center.

and writing the cross-product as

$$\mathbf{x} \times \mathbf{y} = [\mathbf{x}]_{\times}\mathbf{y}. \tag{2.60}$$

Central Projection Let P be a 3×4 matrix. A *central projection*, p, is a linear mapping from homogeneous 4-vectors to homogeneous 3-vectors defined as $p(\mathbf{X}) = \mathrm{P}\mathbf{X}$. The matrix P is known as the *projection matrix*.

The Basic Pinhole Camera

We assume that cameras follow the rules of the central projection, resulting in the pinhole model, shown in Figure 2.8. Furthermore, we assume the center of projection is at $\mathbf{C} = (0, 0, 0)^T$ and that the *image plane* is the plane $Z = f$. The point $\mathbf{C}$ is called the *camera center* or *optical center*. The line that passes through the camera center and is perpendicular to the image plane is called the *principal axis*. The point $\mathbf{p} = (0, 0, f)^T$, at which the principal axis intersects the image plane is called the *principal point*. Under the central projection, a point in space with coordinates $\mathbf{X} = (X, Y, Z)^T$ is mapped to the point where the line joining $\mathbf{C}$ and $\mathbf{X}$ intersects the image plane. We derive the mapping from 3D scene points to 2D image points by considering similar triangles in Figure 2.9. The mapping is given by:

$$(X, Y, Z)^T \mapsto (fX/Z, fY/Z)^T. \tag{2.61}$$

In an actual camera, the image plane lies behind the camera center, but in this

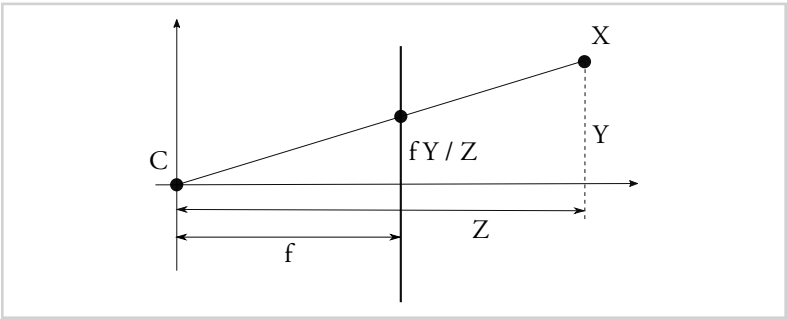


Figure 2.9
Using similar triangles to explain projection.

work—and in computer vision in general—we place the image plane in front of the camera center for mathematical convenience.

Central Projection Using Homogeneous Coordinates

When homogeneous coordinates are used to represent world and image points, the mapping in Equation 2.61 can be expressed as a central projection,

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}. \tag{2.62}$$

The 3×4 projection matrix in Equation 2.62 can be written as the product of two matrices: $\text{diag}(f, f, 1)[I \mid 0]$, where $\text{diag}(f, f, 1)$ is a 3×3 diagonal matrix and $[I \mid 0]$ is a 3×4 matrix composed of a 3×3 identity matrix and a 3×1 vector of zeros.

Let $\mathbf{X} = (X, Y, Z, 1)^T$ denote the world point as a homogeneous 4-vector and let $\mathbf{x}$ denote the image point as a homogeneous 3-vector, then the central projection under the basic pinhole model can be written as

$$\mathbf{x} = \text{diag}(f, f, 1)[I \mid 0]\mathbf{X}. \tag{2.63}$$

Principal Point Offset

It is convenient to move the origin of the image plane away from the principal point so that it corresponds with the lower-left corner of the image. We extend the mapping

from Equation 2.61 and get the following mapping:

$$(X, Y, Z)^T \mapsto (fX/Z + p_x, fY/Z + p_y)^T, \tag{2.64}$$

where $(p_x, p_y)^T$ is the new origin of the image plane. Using homogeneous coordinates, we can express this mapping as a central projection,

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} fX + Zp_x \\ fY + Zp_y \\ Z \end{pmatrix} = \begin{bmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}. \tag{2.65}$$

With the addition of the principal point offset, the projection matrix becomes $P = K[I \mid 0]$, where the 3×3 matrix K, called the *camera calibration matrix*, is defined as

$$K = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.66}$$

The pinhole model with the addition of the principal point offset is, therefore, defined as $\mathbf{x} = K[I \mid 0]\mathbf{X}$.

Camera Rotation and Translation

We have, thus far, assumed that the camera is located at the origin of a Euclidean coordinate system with the principal coordinate pointing in the direction of the z-axis. We call this coordinate system the *camera coordinate frame*. In general this assumption will not hold and points in space will be represented in terms of a different coordinate system, known as the *world coordinate frame*. The two coordinate frames are related by a rotation and a translation:

$$\tilde{\mathbf{X}}_{\text{cam}} = R(\tilde{\mathbf{X}} - \tilde{C}), \tag{2.67}$$

where $\tilde{\mathbf{X}}$ is a point in the world coordinate frame, $\tilde{\mathbf{X}}_{\text{cam}}$ is the same point in the camera coordinate frame, R is a 3×3 rotation matrix encoding the orientation of the camera relative to the world coordinate frame and $\tilde{C}$ is the camera center in the world coordinate frame. Vectors $\tilde{\mathbf{X}}_{\text{cam}}$, $\tilde{\mathbf{X}}$, and $\tilde{C}$ are represented using inhomogeneous 3-vectors as indicated by the tilde sign above their names. Equation 2.67 can also be written in

homogeneous coordinates:

$$\mathbf{X}_{\text{cam}} = \begin{bmatrix} R & -R\tilde{C} \\ 0 & 1 \end{bmatrix} \mathbf{X}, \tag{2.68}$$

and the projection is defined as

$$\mathbf{x} = K[I \mid 0]\mathbf{X}_{\text{cam}} = KR[I \mid -\tilde{C}]\mathbf{X}. \tag{2.69}$$

The three parameters contained in matrix K are called the *internal camera parameters*, while the parameters in R and $\tilde{C}$ which give the position and orientation of the camera relative to the world coordinate frame are called *external camera parameters*.

Digital Camera

Since digital cameras may have rectangular, rather than square, pixels we need to adjust our mathematical model accordingly. Let m_x and m_y be the number of pixels per unit distance in the x and y direction, respectively. To account for the uneven pixel scaling, matrix K, defined in Equation 2.66, takes the form:

$$K = \begin{bmatrix} \alpha_x & 0 & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.70}$$

where $\alpha_x = fm_x$ and $\alpha_y = fm_y$ represent the focal lengths of the digital camera in pixels. Likewise, $x_0 = m_x p_x$ and $y_0 = m_y p_y$ represent the camera center in pixels.

Summary

The mapping from world points to image points is modeled with the central projection. Let $\mathbf{X}$ be homogeneous 4-vector, $\mathbf{x}$ be a homogeneous 3-vector, and let P be a 3×4 projection matrix. In the process of image formation, the world point $\mathbf{X}$ is mapped to the image points $\mathbf{x}$ in the following way:

$$\mathbf{x} = \text{P}\mathbf{X} = KR[I \mid -\tilde{C}]\mathbf{X} = \begin{bmatrix} \alpha_x & 0 & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} R[I \mid -\tilde{C}]\mathbf{X}. \tag{2.71}$$

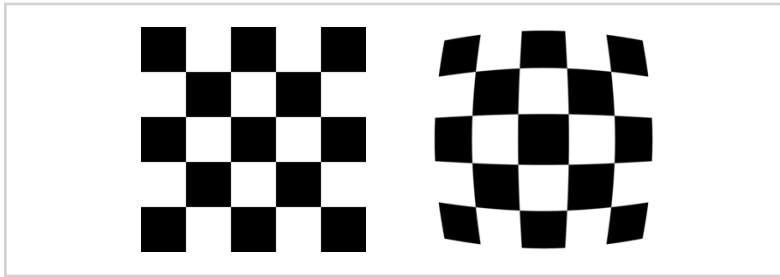


Figure 2.10

Barrel distortion is a lens effect that causes an image to appear as if it has been wrapped around a sphere.

Lens Distortion

Using an actual pinhole camera is not the best way to take images because the amount of light that passes through the pinhole is small. A lens is typically used in place of the pinhole, which increases the amount of light reaching the film. The downside of using a lens is that it introduces distortions and the pinhole camera model is no longer an accurate model of the image formation process.

The most important deviation from the pinhole model is caused by *radial distortion*, where pixels near the boarder get distorted, causing a barrel effect as seen in Figure 2.10. The distortion is zero near the center of the image and increases towards the edges. Let (x, y) denote the image location of a 3D point $\mathbf{X}$ under the pinhole model and (x_d, y_d) denote the image location of the same 3D point $\mathbf{X}$ with radial distortion. The relation between (x, y) and (x_d, y_d) is described with the following equation:

$$\begin{pmatrix} x_d \\ y_d \end{pmatrix} = \begin{pmatrix} x_c \\ y_c \end{pmatrix} + L(r) \begin{pmatrix} x - x_c \\ y - y_c \end{pmatrix}, \tag{2.72}$$

where (x_c, y_c) denotes the center of the radial distortion; r denotes the euclidean distance of (x, y) from the center of distortion, that is, $r^2 = (x - x_c)^2 + (y - y_c)^2$; and $L(r)$ is typically defined as $L(r) = 1 + \kappa_1 r + \kappa_2 r^2 + \kappa_3 r^3$. The coefficients κ_1, κ_2, and κ_3 are determined—together with other internal camera parameters—in the process of image calibration.

2.2.2 *Epipolar Geometry*

The *epipolar geometry* is the intrinsic projective geometry between two views and depends only on the cameras' internal parameters and their relative position and orien-

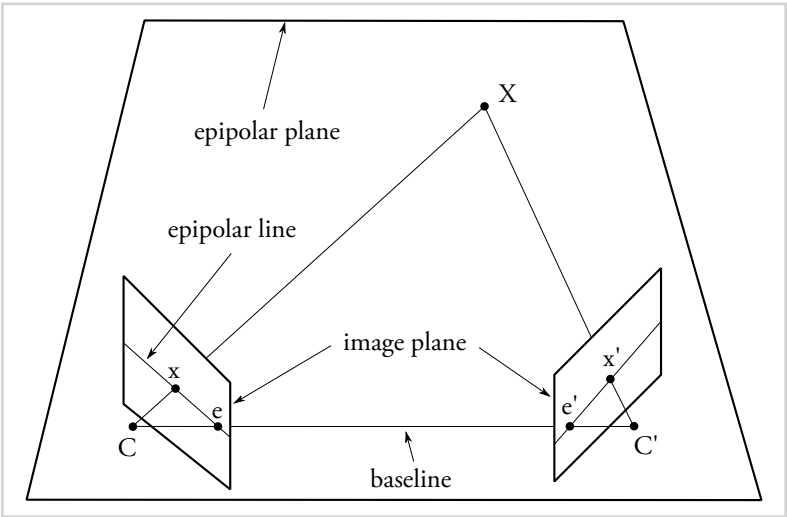


Figure 2.11

The camera centers **C** and **C′**, the point **X** and its two images **x** and **x′** lie on the same plane, called the *epipolar plane*.

tation. It is the geometry of the intersection of the image planes with the set of planes containing the baseline, known as the *epipolar planes*. Studying epipolar geometry is of great importance for solving the stereo correspondence problem, as it limits the search for correspondences to a one dimensional search along a line.

Two cameras, with camera centers **C** and **C′**, are observing a scene point **X**. Let **x** be the image of point **X** in the first camera and **x′** be the image of point **X** in the second camera. We would like to know how **x** and **x′** are related. As shown in Figure 2.11, all five points—the camera centers **C** and **C′**, the scene point **X** and its two images **x** and **x′**—lie on the same plane, known as the *epipolar plane*. The point at which the baseline intersects the image plane is called the *epipole*, denoted as **e** on the left and **e′** on the right image plane.

Consider the epipolar plane defined by **x**, **C** and **C′**. We know that **x′** lies on the same epipolar plane. The point **x′** is, therefore, constrained to lie on the intersection of the epipolar plane and the image plane. The intersection of these two planes is a line, known as the *epipolar line*. This result is important and worth stating again. We assume that the internal camera parameters and the cameras' relative position and orientation are known. Given a point **x** in the first view we wish to find its corresponding point

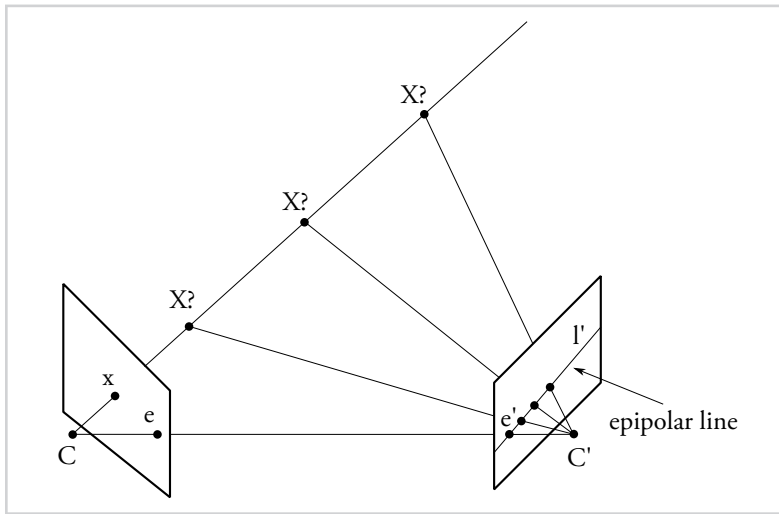


Figure 2.12

The epipolar line in the second view $\mathbf{l}'$ is defined as the intersection of the second image plane and the epipolar plane, which is itself defined by $\mathbf{x}$. This figure illustrates that, given point $\mathbf{x}$ its corresponding point $\mathbf{x}'$ has to lie on the epipolar line $\mathbf{l}'$.

$\mathbf{x}'$ in the second view. We have just shown that point $\mathbf{x}'$ is constrained to lie on the epipolar line $\mathbf{l}'$ that depends only on $\mathbf{x}$.

Another way to see this is illustrated in Figure 2.12. Given an image point $\mathbf{x}$, the scene point $\mathbf{X}$ is constrained to lie on the line that passes through $\mathbf{C}$ and $\mathbf{x}$. If we project points from that line onto the second image plane, we observe that they all lie on the epipolar line $\mathbf{l}'$. In fact, we can define the epipolar line $\mathbf{l}'$ as the projection of the line that passes through $\mathbf{C}$ and $\mathbf{x}$ onto the second image plane.

An example of epipolar geometry on a pair of images is given in Figure 2.13.

2.2.3 *The Fundamental Matrix*

In this section we introduce the fundamental matrix $\mathtt{F}$ that describes the mapping from image points to epipolar lines. Consider the following two points: (1) $\mathbf{C}$, the first camera center, and (2) $\mathtt{P}^{+}\mathbf{x}$, where $\mathbf{x}$ is a point on the image plane of the first camera, $\mathtt{P}$ is the projection matrix of the first camera, and $\mathtt{P}^{+}$ denotes its pseudo-inverse, that is $\mathtt{P}\mathtt{P}^{+} = \mathtt{I}$. Both points lie on the line that passes through $\mathbf{C}$ and $\mathbf{x}$. The first point, $\mathbf{C}$, does so by definition; the second, $\mathtt{P}^{+}\mathbf{x}$, because it projects to the point $\mathbf{x}$, that is, $\mathtt{P}\mathtt{P}^{+}\mathbf{x} = \mathbf{x}$.

The projections of these two points in the second view lie on the epipolar line $\mathbf{l}'$. Let

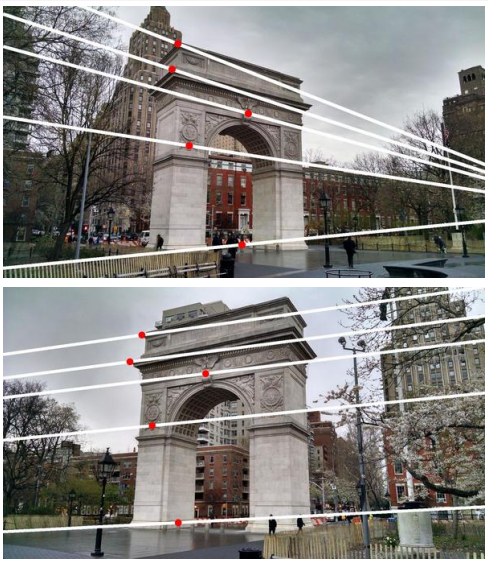


Figure 2.13

Two views of the Washington Square Arch with epipolar lines drawn through five corresponding points.

$\mathbf{P}'$ be the projection matrix of the second camera, then the two points are projected in the second view as $\mathbf{P}'\mathbf{C}$ and $\mathbf{P}'\mathbf{P}^{+}\mathbf{x}$. Note that $\mathbf{P}'\mathbf{C}$ is equal to $\mathbf{e}'$—the epipole—because it is the image of the first camera center in the second view. The epipolar line $\mathbf{l}'$ is defined as

$$\mathbf{l}' = (\mathbf{P}'\mathbf{C}) \times (\mathbf{P}'\mathbf{P}^{+}\mathbf{x}) = \mathbf{e}' \times (\mathbf{P}'\mathbf{P}^{+}\mathbf{x}). \tag{2.73}$$

Recall, from Equation 2.58, that the line $\mathbf{l}$ joining points $\mathbf{x}$ and $\mathbf{y}$ is defined as the cross product $\mathbf{l} = \mathbf{x} \times \mathbf{y}$. We can write Equation 2.73 as a single matrix-vector multiplication by expanding the epipole $\mathbf{e}'$ into the 3×3 matrix $[\mathbf{e}']_{\times}$ (see Equation 2.60). By doing so, we derive $\mathbf{l}' = [\mathbf{e}']_{\times}\mathbf{P}'\mathbf{P}^{+}\mathbf{x}$ or

$$\mathbf{l}' = \mathbf{F}\mathbf{x}, \tag{2.74}$$

where $\mathbf{F} = [\mathbf{e}']_{\times}\mathbf{P}'\mathbf{P}^{+}$ is the *fundamental matrix*—the 3×3 matrix that maps points to epipolar lines.

The fundamental matrix is homogeneous:

$$\mathbf{F} \sim k\mathbf{F} \text{ for } k \neq 0. \tag{2.75}$$

That is, $\mathtt{F}$ and $k\mathtt{F}$ are equivalent; they both produce the same mapping from points to lines. This is true because $\mathtt{F}\mathbf{x} = \mathbf{l}'$, $k\mathtt{F}\mathbf{x} = k\mathbf{l}'$, and since lines are given as homogeneous 3-vectors, lines $\mathbf{l}'$ and $k\mathbf{l}'$ represent the same line.

Correspondence Condition

Because point $\mathbf{x}'$ lies on the epipolar line $\mathbf{l}' = \mathtt{F}\mathbf{x}$, it follows from Equation 2.57 that

$$\mathbf{x}'^T \mathtt{F}\mathbf{x} = 0, \tag{2.76}$$

that is, points $\mathbf{x}$ and $\mathbf{x}'$ correspond if and only if $\mathbf{x}'^T \mathtt{F}\mathbf{x} = 0$. Equation 2.76 characterizes the fundamental matrix $\mathtt{F}$ in terms of point correspondences, without reference to the camera projection matrices $\mathtt{P}$ and $\mathtt{P}'$. This enables the fundamental matrix $\mathtt{F}$ to be estimated from image correspondences alone.

Fundamental Matrix for a Calibrated Camera Stereo Rig

Suppose we have two cameras with the following projection matrices: $\mathtt{P} = \mathtt{K}[\mathtt{I} \mid \mathbf{0}]$ and $\mathtt{P}' = \mathtt{K}'[\mathtt{R} \mid \mathbf{t}]$; that is, the first camera center is located at the world origin with its principal ray equal to the z-axis. The camera center $\mathbf{C}$ and the pseudo-inverse of the projection matrix $\mathtt{P}^+$ are defined as

$$\mathbf{C} = \begin{pmatrix} \mathbf{0} \\ 1 \end{pmatrix} \text{ and } \mathtt{P}^+ = \begin{bmatrix} \mathtt{K}^{-1} \\ \mathbf{0}^T \end{bmatrix}.$$

We can write the fundamental matrix $\mathtt{F}$ in terms of the cameras' internal parameters and their relative position and orientation as

$$\mathtt{F} = [\mathbf{e}']_\times \mathtt{P}'\mathtt{P}^+ = [\mathtt{P}'\mathbf{C}]_\times \mathtt{P}'\mathtt{P}^+ = [\mathtt{K}'\mathbf{t}]_\times \mathtt{K}'\mathtt{R}\mathtt{K}^{-1} \tag{2.77}$$

Fundamental Matrix for a Parallel Camera Stereo Rig

Assume the cameras of a stereo rig have identical internal parameters, that is, $\mathtt{K}' = \mathtt{K} = \text{diag}(f, f, 1)$; both cameras axis are aligned with the z-axis, that is, $\mathtt{R} = \mathtt{I}$; and the camera centers differ only in the x-coordinate, that is, $\mathbf{t} = (t_x, 0, 0)^T$. The fundamental

matrix $\mathtt{F}$ can, then, be expressed as

$$\mathtt{F} = [\mathtt{K}'\mathbf{t}]_\times \mathtt{K}'\mathtt{R}\mathtt{K}^{-1} = [\mathtt{K}\mathbf{t}]_\times \mathtt{K}\mathtt{I}\mathtt{K}^{-1} = [\mathtt{K}\mathbf{t}]_\times = \\ = \begin{bmatrix} ft_x \\ 0 \\ 0 \end{bmatrix}_\times = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -ft_x \\ 0 & ft_x & 0 \end{bmatrix} \sim \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}.$$

The last step of the derivation follows from Equation 2.75.

To see how this special form of the fundamental matrix restricts the search for correspondences we expand Equation 2.76.

$$\mathbf{x}'^T \mathtt{F}\mathbf{x} = 0$$

$$\begin{pmatrix} x' & y' & 1 \end{pmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = 0$$

$$-y' + y = 0$$

$$y = y'.$$

If points $\mathbf{x} = (x, y)^T$ and $\mathbf{x}' = (x', y')^T$ correspond it follows that $y = y'$, or, in other words, the epipolar line of $\mathbf{x} = (x, y)^T$ is the horizontal line $y' = y$. Aligning the epipolar lines in this way simplifies the implementation of stereo algorithms, because the search for corresponding points is carried out along horizontal lines. Unfortunately, constructing a perfect parallel stereo rig is nearly impossible in practice, but we can correct for the mechanical misalignments by transforming the two views in a process known as *image rectification*.

Image rectification is the process of resampling pairs of stereo images so that all epipolar lines are horizontal and corresponding points have identical y-coordinates. A pair of projective transformations are applied to the two images in order to match the epipolar lines. Since there are many projective transformation that achieve this, a typical rectification method attempts to find a pair of transformations that subject the images to minimal distortion. After image rectification, the epipolar geometry is the same as if the images were taken from a parallel stereo rig. Because most stereo methods assume this to be the case, image rectification is often performed before the stereo method.

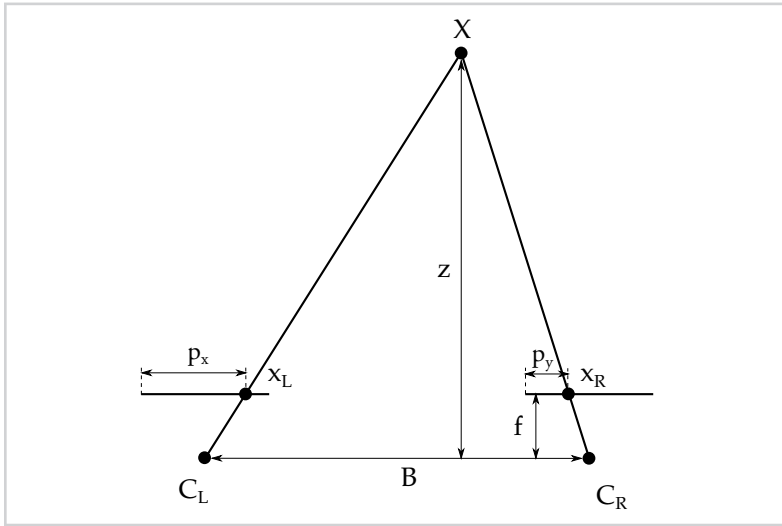


Figure 2.14
Depth and disparity are inversely proportional.

Depth and Disparity

Depth and disparity are inversely proportional as we saw in Equation 1.1. We derive this relation in this section using similar triangles.

Consider a parallel camera stereo rig shown in Figure 2.14. C_L and C_R denote the camera centers, X denotes a 3D scene point that projects to x_L and x_R in the left and right image planes. Let f denote the focal length, B the baseline, d the disparity and z the depth of point X. The disparity d is defined as the difference in horizontal location of x_L in the left and x_R in the right image, that is, $d = p_x - p_y$. Triangles $\Delta C_L X C_R$ and $\Delta x_L X x_R$ are similar triangles, therefore:

$$\frac{B}{z} = \frac{B-d}{z-f} \implies z = \frac{fB}{d}, \tag{2.78}$$

which proves Equation 1.1.

2.3 Stereo Vision

Early work on computational stereo started in the 1970s by the Image Understanding community. Barnard and Fischler [6] reviewed stereo research up to 1981. Their pa-

per focuses on the fundamentals of computational stereo (image acquisition, camera modeling, feature acquisition, matching, distance determination, and interpolation), evaluation criteria, and contains a survey of a representative sampling of the Image Understanding work relevant to computational stereo. Work on stereo continued in the 1980s, with a review paper published in 1989 by Dhond and Aggarwal [36], which surveys many new stereo matching methods. It also introduces hierarchical processing and the use of trinocular constraints. The two most recent review papers on the topic are Brown et al. [22] from 2003, which focuses on correspondence methods, methods for dealing with occlusion, and real-time implementations; and Scharstein and Szeliski [113] from 2002, which provides a taxonomy and categorization of existing stereo algorithms allowing individual components of the stereo algorithm to be compared. A good overview of stereo correspondence algorithms is also given in the book by Szeliski [131].

In this section, we focus on methods that operate on two rectified images and produce a dense disparity map. According to Scharstein and Szeliski [113] stereo algorithms perform a subset of the following four steps: (1) matching cost computation, (2) cost aggregation, (3) disparity computation, and (4) disparity refinement. To see how a stereo algorithm decomposes into the four steps, consider a simple stereo algorithm with a *sum of absolute differences* matching cost and a winner-takes-all strategy. (1) The matching cost is computed as the absolute difference between the intensity values of a pixel in the left and a pixel in the right image. (2) The cost is aggregated by computing the average matching cost within a small rectangular window of neighbouring pixels. (3) The disparity is computed by selecting the disparity that minimizes the aggregated cost.

Some methods are defined on small image patches and perform steps (1) and (2) jointly. Examples include normalized cross-correlation [16, 59], the rank and census transforms Zabih and Woodfill [152], and our method based on convolutional neural networks [154].

Methods which compute disparity by considering only a small neighbourhood around the pixel of interest are know as *local* algorithms. *Global* algorithms, on the other hand, take the whole image into account, typically by making explicit smoothness assumptions and solving an optimization problem. Global algorithms usually omit the cost aggregation step as the information from neighbouring pixels is propagated during the optimization step. Global methods define a cost function, which usually contains a

data term and a smoothness term. Common optimization algorithms include iterated conditional modes [12], simulated annealing [5, 76, 94], probabilistic diffusion [112], graph cuts [18, 78], and message passing algorithms [77, 143, 150]. Soma algorithms are difficult to classify as either local or global. Certain iterative algorithms, for example, do not explicitly state their global cost function, but their iterative behaviour is similar to some optimization algorithms [93, 112, 160]. Hierarchical algorithms resemble such iterative algorithms. They operate on a pyramid of images, solving the correspondence problem at a coarse level first and refine their solution at each successive step [9, 37, 38, 95, 106, 147].

2.3.1 Matching Costs

Since the main contribution of this thesis is a new matching cost, we review existing matching cost functions in great detail.

A matching cost $C(\mathbf{p}, d)$ is a function that maps a position $\mathbf{p}$ and a disparity d into a real number, which is interpreted as the cost of matching position $\mathbf{p}$ in the left image with position $\mathbf{p} - \mathbf{d}$ in the right image. We want the cost to be lowest when the left and right positions correspond. We use bold lowercase letters $\mathbf{p}$ and $\mathbf{q}$ to denote image locations. A bold lowercase $\mathbf{d}$ denotes the disparity d cast to a vector, that is, $\mathbf{d} = (d, 0)$.

Simple matching costs assume constant intensities at matched locations, but more advanced matching costs are more robust and can compensate for certain radiometric differences and noise. Radiometric differences are caused, for example, due to slightly different camera settings, vignetting, and image noise. Reducing the stereo baseline would reduce these differences, but would also reduce the geometric accuracy of triangulation. The following section describes several matching costs as they were presented by Hirschmüller and Scharstein [64]. They group the matching costs into three categories: parametric matching costs, non-parametric matching costs, and mutual information.

Parametric Matching Costs

Parametric costs use the magnitude of the pixel intensities.

Absolute differences Probably the simplest parametric matching cost is the *absolute differences*, which assumes brightness consistency for corresponding pixels, that is, it

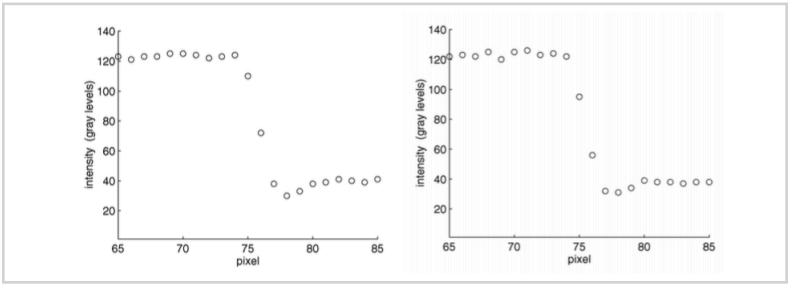


Figure 2.15

Two scanlines differing by a disparity of 0.4. Image taken from [14].

assumes the intensity values of corresponding pixels are the same. The absolute differences matching cost C_{AD} is defined as

$$C_{\text{AD}}(\mathbf{p}, d) = |I^L(\mathbf{p}) - I^R(\mathbf{p} - \mathbf{d})| \tag{2.79}$$

where $I^L(\mathbf{p})$ and $I^R(\mathbf{p})$ are image intensities at position $\mathbf{p}$ in the left and right image, respectively.

Sum of absolute differences Local stereo methods typically aggregate this matching cost over all pixels within a local neighbourhood. This is referred to as the sum of absolute differences matching cost C_{SAD} and is defined as

$$C_{\text{SAD}}(\mathbf{p}, d) = \sum_{\mathbf{q} \in N_{\mathbf{p}}} |I^L(\mathbf{q}) - I^R(\mathbf{q} - \mathbf{d})|, \tag{2.80}$$

where $N_{\mathbf{p}}$ is the set of locations within a rectangular window centered at $\mathbf{p}$.

Birchfield-Tomasi Image sampling can cause traditional matching costs to assign high values to a corresponding pair of pixels even in the absence of noise. Birchfield and Tomasi [14] propose a method for measuring the matching cost that is insensitive to image sampling.

Let i^L and i^R be one-dimensional continuous intensity functions. The functions i^L and i^R are sampled at discrete points by the image sensors, resulting in two discrete one-dimensional arrays of image intensity values I^L and I^R, as shown in Figure 2.15. Let $\hat{I}^L$ and $\hat{I}^R$ denote the linearly interpolated functions between the sample points of

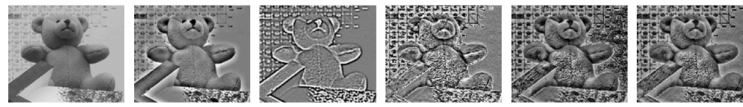


Figure 2.16
Different filters on the Teddy stereo image pair. Image taken from [64].

I^L and I^R. The Birchfield-Tomasi cost function is defined as

$$C_{\text{BT}}(\mathbf{p}, d) = \min\Big\{ \min_{-0.5 \le d' \le 0.5} I^L(\mathbf{p}) - \hat{I}^R(\mathbf{p} - \mathbf{d} + (d', 0)^T), \\ \min_{-0.5 \le d' \le 0.5} \hat{I}^L(\mathbf{p} + (d', 0)^T) - I^R(\mathbf{p} - \mathbf{d}) \Big\}. \tag{2.81}$$

Since the extreme points of a piecewise linear function must be its breakpoints, the computation of the Birchfield-Tomasi cost can be computed efficiently using the following equations:

$$\begin{aligned} C_{\text{BT}}(\mathbf{p}, d) &= \min(A, B) \\ A &= \max(0, I^L(\mathbf{p}) - I^R_{\max}(\mathbf{p} - \mathbf{d}), I^R_{\min}(\mathbf{p} - \mathbf{d}) - I^L(\mathbf{p})) \\ B &= \max(0, I^R(\mathbf{p} - \mathbf{d}) - I^L_{\max}(\mathbf{p}), I^L_{\min}(\mathbf{p}) - I^R(\mathbf{p} - \mathbf{d})) \\ I_{\min}(\mathbf{p}) &= \min(I_-(\mathbf{p}), I(\mathbf{p}), I_+(\mathbf{p})) \\ I_{\max}(\mathbf{p}) &= \max(I_-(\mathbf{p}), I(\mathbf{p}), I_+(\mathbf{p})) \\ I_-(\mathbf{p}) &= (I(\mathbf{p} - (1, 0)^T) + I(\mathbf{p}))/2 \\ I_+(\mathbf{p}) &= (I(\mathbf{p} + (1, 0)^T) + I(\mathbf{p}))/2 \end{aligned} \tag{2.82}$$

Another window-based sampling-insensitive cost function was developed by Čech and Šára [24].

The next three matching costs are filters that are applied to the two images separately before computing the matching cost with absolute differences.

Mean filter The *mean filter* subtracts, at each location of the image, the mean of the intensity values within a square window centered at the pixel of interest. A constant offset of 128 is usually added to avoid negative numbers when using unsigned 8-bit variables. The mean filter is defined as

$$I_{\text{mean}}(\mathbf{p}) = I(\mathbf{p}) - \frac{1}{|N_\mathbf{p}|} \sum_{\mathbf{q} \in N_\mathbf{p}} I(\mathbf{q}) + 128. \tag{2.83}$$

Laplacian of Gaussian The *Laplacian of Gaussian* (LoG) filter performs smoothing for removing noise and removes an offset in intensities. It is computed by convolving the image with a LoG kernel,

$$I_{\text{LoG}} = I * K_{\text{LoG}} \tag{2.84}$$
$$K_{\text{LoG}}(x,y) = -\frac{1}{\pi\sigma^4}\left(1 - \frac{x^2+y^2}{2\sigma^2}\right)e^{\frac{x^2+y^2}{2\sigma^2}},$$

where σ is the standard deviation. The LoG filter is often used in real-time methods [65, 81].

Background subtraction by bilateral filtering The final filter we consider is *background subtraction by bilateral filtering* (BilSub) developed by Ansar et al. [4]. The bilateral filter [134] has the effect of a gaussian blur without blurring high contrast texture. It sums intensity values of neighboring pixels weighted according to proximity and color similarity. The BilSub filter is computed by subtracting, at each position, the value of the bilateral filter at that location:

$$I_{\text{BilSub}}(\mathbf{p}) = I(\mathbf{p}) - \frac{\sum_{\mathbf{q}\in N_\mathbf{q}} I(\mathbf{p})e^s e^r}{\sum_{\mathbf{q}\in N_\mathbf{q}} e^s e^r}, \tag{2.85}$$
$$s = -\frac{\|\mathbf{q}-\mathbf{p}\|}{2\sigma_s^2},$$
$$r = -\frac{(I(\mathbf{q})-I(\mathbf{p}))^2}{2\sigma_r^2}.$$

The parameter σ_s controls the amount of smoothing, while the parameter σ_r prevents smoothing over high-contrast regions.

Zero-mean sum of absolute differences The *zero-mean sum of absolute differences* (ZSAD) subtracts the mean intensity value of a window from every pixel in that windows and computes the sum of absolute differences.

$$C_{ZSAD}(\mathbf{p},d) = \sum_{\mathbf{q}\in N_\mathbf{p}} |I^L(\mathbf{q}) - \bar{I}^L(\mathbf{p}) - I^R(\mathbf{q}-\mathbf{d}) + \bar{I}^R(\mathbf{p}-\mathbf{d})| \tag{2.86}$$
$$\bar{I}(\mathbf{p}) = \frac{1}{N_\mathbf{p}} \sum_{\mathbf{q}\in N_\mathbf{p}} I(\mathbf{q})$$

In zero-mean sum of absolute differences the same mean value is subtracted at every location in the window. Note that this is similar, but not quite the same, as applying the mean filter from Equation 2.83 followed by the sum of absolute differences cost, where the mean is computed for every pixel individually.

Normalized cross-correlation Normalized cross-correlation (NCC) is a window-based method that is commonly used in practice. It can compensate for gain changes but tends to work poorly on depth discontinuities, because outliers are strongly penalized. The normalized cross-correlation C_{NCC} is defined as:

$$C_{NCC}(\mathbf{p}, d) = \frac{\sum_{\mathbf{q} \in N_\mathbf{p}} I^L(\mathbf{q}) I^R(\mathbf{q} - \mathbf{d})}{\sqrt{\sum_{\mathbf{q} \in N_\mathbf{p}} I^L(\mathbf{q})^2 \sum_{\mathbf{q} \in N_\mathbf{p}} I^R(\mathbf{q} - \mathbf{d})^2}} \tag{2.87}$$

A variant of normalized cross-correlation was presented by Morevec [99]. It only approximates Equation 2.87, but is faster to compute.

Zero-mean normalized cross-correlation The last parametric matching cost is the zero-mean normalized cross-correlation (ZNCC). It is similar to NCC but can additionally compensate for differences in offset within the correlation window. The zero-mean normalized cross-correlation cost C_{ZNCC} is defined as:

$$C_{ZNCC}(\mathbf{p}, d) = \frac{\sum_{\mathbf{q} \in N_\mathbf{p}} (I^L(\mathbf{q}) - \bar{I}^L(\mathbf{p}))(I^R(\mathbf{q} - \mathbf{d}) - \bar{I}^R(\mathbf{p} - \mathbf{d}))}{\sqrt{\sum_{\mathbf{q} \in N_\mathbf{p}} (I^L(\mathbf{q}) - \bar{I}^L(\mathbf{p}))^2 \sum_{\mathbf{q} \in N_\mathbf{p}} (I^R(\mathbf{q} - \mathbf{d}) - \bar{I}^R(\mathbf{p} - \mathbf{d}))^2}} \tag{2.88}$$

Non-parametric Matching Costs

Whereas parametric matching costs used pixel intensity values, non-parametric matching costs use only their local ordering. Most of them can be implemented as filters.

Rank filter The Rank filter is defined with the following equation:

$$I_{\text{Rank}}(\mathbf{p}) = \sum_{\mathbf{q} \in N_\mathbf{p}} 1\{I(\mathbf{q}) < I(\mathbf{p})\}, \tag{2.89}$$

where $1\{\cdot\}$ denotes the indicator function. The Rank filter at position $\mathbf{p}$ is defined as the number of positions $\mathbf{q}$ in some region of interest that have a smaller intensity value than $\mathbf{p}$. It was proposed by Zabih and Woodfill [152] to increase robustness to outliers

within the region of interest. Outliers frequently occur near depth discontinuities. Since the rank filter only depends on the ordering of image intensity values and not on the actual values, it is invariant to all transformations that preserve the ordering. When computing the matching cost, the rank-transformed images are compared using the absolute differences method.

Soft Rank filter Since the Rank filter uses only the ordering of intensity values, it is sensitive in areas with low texture. Instead of always making hard decision when comparing image intensity values as in Equation 2.89, it might seem reasonable to soften the decision when positions $\mathbf{p}$ and $\mathbf{q}$ have similar image intensity values. The Soft Rank filter defines a linear—soft transition—zone between 0 and 1 for similar intensity values. It is defined with the following equation:

$$I_{\text{SoftRank}}(\mathbf{p}) = \sum_{\mathbf{q} \in N_{\mathbf{p}}} \min\left(1, \max\left(0, \frac{I(\mathbf{p}) - I(\mathbf{q})}{2t} + \frac{1}{2}\right)\right), \tag{2.90}$$

where t is the hyperparameter of the filter that controls the size of the transition zone.

Census filter The census filter [152] represents each image position as a bit vector. The vector is computed by cropping an image patch centered around the position of interest and comparing the intensity values of each pixel in the patch to the intensity value of the pixel in the center. When the center pixel is darker the corresponding bit is set. Let D be the set of displacements and let $\bigotimes$ denote concatenation. The census transformed is defined as

$$I_{\text{Census}}(\mathbf{p}) = \bigotimes_{[i,j] \in D} 1\{I(\mathbf{p}) < I(\mathbf{p} + (i,j)^T)\}. \tag{2.91}$$

The matching cost is defined as the hamming distance between two census transformed vectors. The performance of the Census filter is better than the Rank filter, but the computation costs are higher due to the calculation of the hamming distance [152].

Ordinal measure The ordinal measure was proposed by Bhat et al. [13]. It is based on the distance of rank permutations of corresponding matching windows. It can't be implemented as a filter and requires window-based matching.

Mutual Information

Mutual information was popularized in computer vision by Viola and Wells [139]. It is primarily used for image registration. Mutual information of images I_1 and I_2 is defined as the sum of entropies of the individual images, H_{I_1} and H_{I_2}, minus their joint entropy, H_{I_1,I_2}:

$$\mathrm{MI}_{I_1,I_2} = H_{I_1} + H_{I_2} - H_{I_1,I_2}. \tag{2.92}$$

The entropies are computed from probability distributions P_{I_1}, P_{I_2}, and P_{I_1,I_2}, which are derived by normalizing the histograms of image intensity values.

$$H_I = -\int_0^1 P_I(i) \log P_I(i)\, di, \tag{2.93}$$

$$H_{I_1,I_2} = -\int_0^1 \int_0^1 P_{I_1,I_2}(i_1, i_2) \log P_{I_1,I_2}(i_1, i_2)\, di_1\, di_2. \tag{2.94}$$

The mutual information of two images will be lower when they are well registered than when they are not. The entropies H_{I_1} and H_{I_2} will not change, but the joint entropy H_{I_1,I_2} will be lower when the images are well registered.

If small image patches are used to compute mutual information there might not be enough pixels to reliably estimate the probability distribution. If, however, large image patches are used, the disparity maps will not be accurate around depth discontinuities. There is a way to compute mutual information based on the whole image and to allow pixel-wise matching [62, 75]. Kim et al. [75] suggest an iterative algorithm that begins with an initial disparity map and refines it at each step. The initial disparity map is either random or is obtained from another stereo matching cost. The initial disparity map is used to warp the second image. We first compute the joint entropy, defined in Equation 2.94, as a sum of terms over all image positions $\mathbf{p}$.

$$H_{I_1,I_2} = \sum_{\mathbf{p}} h_{I_1,I_2}(I_1(\mathbf{p}), I_2(\mathbf{p})), \tag{2.95}$$

$$h_{I_1,I_2}(i,k) = -\frac{1}{n} \log(P_{I_1,I_2}(i,k) * g(i,k)) * g(i,k), \tag{2.96}$$

where n denotes the number of pixels in the image and $*g(\cdot)$ denotes convolution with a gaussian filter. See Kim et al. [75] for the derivation of this step. The probability

distribution P_{I_1,I_2} is defined as

$$P_{I_1,I_2}(i,k) = \frac{1}{n}\sum_{\mathbf{p}} 1\{(i,k) = (I_1(\mathbf{p}), I_2(\mathbf{p}))\}. \tag{2.97}$$

The entropies of the individual images are computed analogously:

$$H_I = \sum_{\mathbf{p}} h_I(I(\mathbf{p})), \tag{2.98}$$

$$h_I(i) = -\frac{1}{n}\log(P_I(i) * g(i)) * g(i). \tag{2.99}$$

The resulting definition of mutual information is:

$$\mathrm{MI}_{I_1,I_2} = \sum_{\mathbf{p}} \mathrm{mi}_{I_1,I_2}(I_1(\mathbf{p}), I_2(\mathbf{p})), \tag{2.100}$$

$$\mathrm{mi}_{I_1,I_2}(i,k) = h_{I_1}(i) + h_{I_2}(k) - h_{I_1,I_2}(i,k). \tag{2.101}$$

Which leads to the definition of the mutual information matching cost function,

$$C_{\mathrm{MI}}(\mathbf{p}, d) = -\mathrm{mi}_{I^L, f_D(I^R)}(I^L(\mathbf{p}), I^R(\mathbf{p} - \mathbf{d})). \tag{2.102}$$

The method is usually performed in a hierarchical fashion [62], by downscaling the image by a factor of 16. After a number of iterations on the small resolution images, the obtained disparity map is upsampled and used as the initial disparity estimate for matching at $\frac{1}{8}$th of the full resolution. This process is repeated until we reach the full resolution images. The disparity maps obtained at a lower resolution are not used to constrain the search on the higher resolution, but just as initial estimates of the disparity map.

2.3.2 *Stereo Methods*

Many problems in early vision, including stereo, can be viewed as assigning labels to pixels. In the case of stereo, the labels represent the disparity at a particular pixel. Pixel labelling problems are often solved by minimizing an energy function that contains the following two terms: a data term, which penalizes solutions that are inconsistent with local observations, and a smoothness term, which enforces some kind of spatial coherence. This type of energy function can be justified in terms of maximum a posteriori

estimation of a Markov Random Field [12, 50]. Wide spread adoption of energy minimization approaches was slow at first, likely due to inefficient optimization algorithms, such as iterated conditional modes [12] and simulated annealing [5, 76]. With the introduction of powerful energy minimization algorithms, such as graph cuts [18, 78] and loopy belief propagation [143, 150], the popularity of energy minimization approaches grew. This section is modeled after the comparative study of different energy minimization methods by Szeliski et al. [132].

The pixel labeling problem is defined as assigning a label $l_{\mathbf{p}}$ to each pixel $\mathbf{p}$ of an image. The number of pixels in an images is n, and the number of labels is m. The energy function E is written as

$$E = E_d + \lambda E_s, \tag{2.103}$$

where E_d is the data term, E_s is the smoothness term, and λ is a real number. The data term E_d is the sum of a set of per-pixel data costs $d_{\mathbf{p}}(l_{\mathbf{q}})$,

$$E_d = \sum_{\mathbf{p}} d_{\mathbf{p}}(l_{\mathbf{q}}). \tag{2.104}$$

We assume that pixels form a 2D grid. Therefore, a position $\mathbf{p}$ can be written in terms of its coordinates $\mathbf{p} = (i, j)$. Furthermore, we assume the standard 4-connected neighbourhood, where each pixels is connected to its top, bottom, left, and right neighbour, that is, pixels $\mathbf{p} = (i, j)$ and $\mathbf{q} = (s, t)$ are connected if and only if $|i-s|+|j-t| = 1$. Let N denote the set of all neighbouring pixel pairs. We can write the smoothness term E_s as the sum of spatially varying costs $V_{\mathbf{pq}}(l_{\mathbf{p}}, l_{\mathbf{p}})$,

$$E_s = \sum_{\{\mathbf{p},\mathbf{q}\}\in N} V_{\mathbf{pq}}(l_{\mathbf{p}}, l_{\mathbf{q}}). \tag{2.105}$$

where the notation $\{\mathbf{p}, \mathbf{q}\}$ denotes unordered sets.

There are special cases of energy functions that have an algorithm for computing the exact solution in polynomial time, unfortunately, none of them will be useful for the problem of stereo matching. If there are only two labels, the Potts model can be solved exactly by graph cuts. This was first observed in the context of Scheduling by Stone [128] and in was first applied to images by Greig et al. [53]. If the labels are integers starting with 0 and the smoothness cost is an arbitrary convex function, Ishikawa [73] gives a graph cut construction. If $V(\Delta l) = \Delta l$ and the data costs are

convex, an algorithm due to Hochbaum [66] can be used. The NP-hardness result proved by Boykov et al. [18] holds if there are more than two labels, as long as the class of smoothness costs include the Potts model.

Iterated Conditional Modes (ICM)

Iterated conditional modes [12] is a greedy algorithm for find a local minimum of the energy function. It starts with an initial assignment of labels to pixels. The initial assignment can be random. Better results are achieved, however, if the initial assignment respects the data term of the energy function. This is achieved by finding the assignment of labels to pixels with λ in Equation 2.103 set to 0. ICM is extremely sensitive to the initial estimate, especially in high-dimensional spaces with non-convex energies due to the large number of local minima.

ICM works by repeatedly applying the following step: for each pixel, set its label to the one that gives the largest decrease in energy. The updates can be performed asynchronously, with pixels being updated one after the other, or synchronously, with all updates occurring at the same time. The algorithm is guaranteed to converge for the asynchronous version and, in practice, the convergence is rapid. For the synchronous version, however, the convergence is not guaranteed as small oscillations may occur.

Graph Cuts

Graph cuts were introduced by Boykov et al. [17, 18], who show that the problem of minimizing the energy function defined in Equation 2.103 reduces to instances of the maximum flow problem in a graph (or minimum cut, which is related to maximum flow by the min-flow max-cut theorem). Two algorithms are presented by Boykov et al. [18], one based on α-β-swap moves and one based on α-expansion moves. These algorithms rapidly compute a local minimum in the sense that no permitted move produces a labelling with lower energy. The two moves are defined as follows:

- For a pair of labels α and β the α-β-swap move exchange the labels between an arbitrary set of pixels labeled α and an arbitrary set labeled β.
- For a label α the α-expansion move assigns an arbitrary set of pixels the label α.

The criteria for a local minimum with respect to swap moves or expansion moves are stronger than moves from ICM and there are much fewer local minima, especially in

high-dimensional spaces. The α-β-swap move was shown to be applicable to any energy where $V_{\mathbf{pq}}$ is a semi-metric, and the α-expansion move algorithm to any energy where $V_{\mathbf{pq}}$ is a metric. These conditions were later generalized by Kolmogorov and Zabin [78].

Max-Product Loopy Belief Propagation

Belief propagation is a message passing algorithm [103]. When applied to acyclic graphs, it finds the global minimum of the energy function. The same algorithm can be applied to general graphs to give an approximate algorithm. This algorithm is called loopy belief propagation, because the graphs contain cycles or loops. The convergence of loopy belief propagation is not guaranteed as it may get trapped in an infinite loop, switching between labellings. The max-product version of loopy belief propagation is used to find an assignment of labels to pixels, while the sum-product version is used to compute the marginal distributions. A detailed description of the loopy belief propagation algorithm is given by Freeman et al. [43] and Felzenszwalb and Huttenlocher [41].

Semiglobal Matching

Semiglobal matching aims to minimize the global 2D energy function by solving a large number of 1D minimization problems [62]. The actual energy function used is defined as

$$E(D) = \sum_{\mathbf{p}} \Big(C(\mathbf{p}, D(\mathbf{p})) + \sum_{\mathbf{q} \in \mathcal{N}_{\mathbf{p}}} P_1 \cdot 1\{|D(\mathbf{p}) - D(\mathbf{q})| = 1\} + \sum_{\mathbf{q} \in \mathcal{N}_{\mathbf{p}}} P_2 \cdot 1\{|D(\mathbf{p}) - D(\mathbf{q})| > 1\} \Big), \quad (2.106)$$

where P_1 is the penalty when neighboring pixels differ in disparity by one pixel and P_2 is the penalty for all larger differences in disparity. Semiglobal matching calculates $E(D)$ along 1D paths from 8 directions towards each pixel of interest using dynamic programming. The costs of all paths are summed for each pixel and disparity.

2.3.3 Evaluation

This section summarizes the results obtained by Hirschmüller and Scharstein [64]. The validation is performed on six stereo image pairs taken from the Middlebury 2005 data

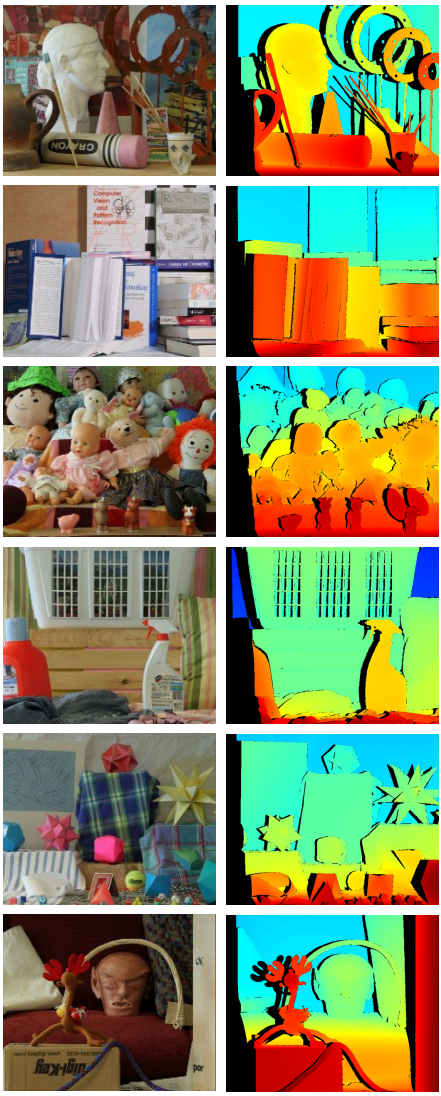


Figure 2.17

The *Art*, *Books*, *Dolls*, *Laundry*, *Moebius*, and *Reindeer* stereo pairs from the Middlebury 2005 data set. The left column contains the reference image and the right column the ground truth disparity map.

set: *Art*, *Books*, *Dolls*, *Laundry*, *Moebius*, and *Reindeer*. See Figure 2.17 . The following matching cost functions were considered:

- Absolute differences (AD)
- Birchfield and Tomasi (BT)
- Mean filter followed by BT (Mean/BT)
- Laplacian of Gaussian followed by BT (LoG/BT)
- Background subtraction by bilateral filtering followed by BT (BilSub/BT)
- Zero-mean sum of absolute differences (ZSAD)
- Normalized cross-correlation (NCC)
- Zero-mean normalized cross-correlation (ZNCC)
- Rank filter followed by AD (Rank/AD)
- Rank filter followed by BT (Rank/BT)
- Soft rank filter followed by AD (SoftRank/AD)
- Soft rank filter followed by BT (SoftRank/BT)
- Census transform (Census)
- Ordinal measure (Ordinal)
- Hierarchical mutual information (HMI)

Some of the matching costs (NCC, ZSAD, ZNCC, and Ordinal) can only be used in window-based matching. Others have been tested with both AD and BT. Results for AD and BT are both reported only when they produced significantly different results.

Since the overall accuracy of a stereo system depends not only on the matching cost but also on the algorithms that uses the cost, the evaluation has to take this into account. It is the interaction of the matching cost and the stereo method that is of interest, because in real-life applications a matching cost is rarely used in isolation and is usually followed by a stereo method. The following three stereo algorithms were considered:

- A local, windows-based method (Window)
- Semiglobal matching (SGM)
- A global method using graph cuts (GC)

Each matching cost was evaluated with all three stereo methods, except for NCC, ZSAD, ZNCC, and Ordinal which can only be used with the local method.

Window The matching cost is aggregated over a square 9×9 window and disparity is determined using the winner-takes-all strategy. The following post-processing steps are used: subpixel interpolation, to obtain disparities with subpixel precision, a left-right consistency check, to detect occlusions and mismatches, and invalidation of disparity segments smaller than 160 pixels. The invalidated regions are filled in by propagating background disparity values from valid nearby regions.

SGM The semiglobal matching algorithm is applied and the disparities are determined using the winner-takes-all strategy. The post-processing steps are similar to the local stereo method: subpixel enhancement, a left-right consistency check, and invalidation of disparity segments smaller than 20 pixels. The invalidated disparities are interpolated as in the local method.

GC The graph cuts algorithm is applied and the disparities are, again, determined using the winner-take-all strategy. Unlike Window and SGM, no post-processing steps are used.

Error is calculated by counting the number of predicted disparities that differ from the ground truth by more than one pixel. Only non-occluded regions of the image are used to determine the error, because occlusions can not be handled by the matching step. The results of the experiments are presented in Figure 2.18.

Many cost functions outperform the sum of absolute differences and Birchfield-Tomasi. It would seem reasonable to expect absolute difference to work best when corresponding points have the same brightness. Even though the images of the Middlebury data set were taken in a studio under controlled conditions the brightness constancy assumption is still violated. To summarize the results of Figure 2.18, the performance of the matching costs can depend on the stereo method used, but Census showed the best performance with all there stereo methods.

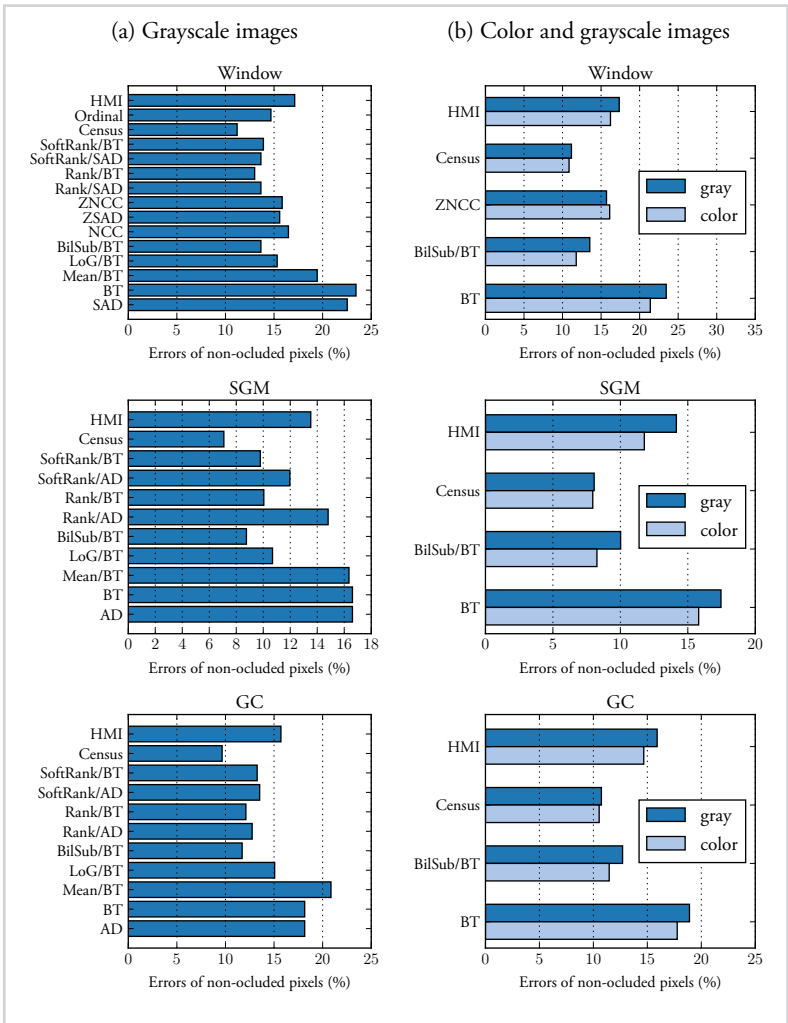


Figure 2.18

Mean 1 pixel errors on the Middlebury 2005 stereo data set. (a) Comparing matching costs on grayscale images. (b) How color information affects the performance of the matching cost.

The images were converted to grayscale before being matched by the stereo algorithm. In many applications, however, color images are available and the additional information might be useful in matching patches. The most promising costs were reimplemented for color images by applying them separately on the red, green, and blue color channels and averaging the results. The results are shown in Figure 2.18 and seem to indicate that using color has little overall benefit.

2.4 Related Work

Before the introduction of large stereo data sets like KITTI and Middlebury, relatively few stereo algorithms used ground truth information to learn parameters of their models; in this section, we review the ones that did.

Kong and Tao [79] used the sum of squared distances to compute an initial matching cost. They then trained a model to predict the probability distribution over three classes: the initial disparity is correct, the initial disparity is incorrect due to fattening of a foreground object, and the initial disparity is incorrect due to other reasons. The predicted probabilities were used to adjust the initial matching cost. Kong and Tao [80] later extend their work by combining predictions obtained by computing normalized cross-correlation over different window sizes and centers. Peris et al. [104] initialized the matching cost with AD-Census [97], and used multiclass linear discriminant analysis to learn a mapping from the computed matching cost to the final disparity.

Ground-truth data was also used to learn parameters of probabilistic graphical models. Zhang and Seitz [158] used an alternative optimization algorithm to estimate optimal values of Markov random field hyperparameters. Scharstein and Pal [111] constructed a new data set of 30 stereo pairs and used it to learn parameters of a conditional random field. Li and Huttenlocher [91] presented a conditional random field model with a non-parametric cost function and used a structured support vector machine to learn the model parameters.

Recent work [57, 126] focused on estimating the confidence of the computed matching cost. Haeusler et al. [57] used a random forest classifier to combine several confidence measures. Similarly, Spyropoulos et al. [126] trained a random forest classifier to predict the confidence of the matching cost and used the predictions as soft constraints in a Markov random field to decrease the error of the stereo method.

A related problem to computing the matching cost is learning local image descriptors [21, 58, 102, 107, 123, 137, 153]. The two problems share a common subtask: to

measure the similarity between image patches. Brown et al. [21] introduced a general framework for learning image descriptors and used Powell's method to select good hyperparameters. Several methods have been suggested for solving the problem of learning local image descriptors, such as boosting [137], convex optimization [123], hierarchical moving-quadrant similarity [107], convolutional kernel networks [102], and convolutional neural networks [58, 153]. Works of Zagoruyko and Komodakis [153] and Han et al. [58], in particular, are very similar to our own, differing mostly in the architecture of the network; concretely, the inclusion of pooling and subsampling to account for larger patch sizes and larger variation in viewpoint.

3

Convolutional Neural Networks for Stereo

In this section we describe a new algorithm for computing the stereo matching cost and show that it outperforms all previously published approaches on the KITTI 2012, KITTI 2015, and Middlebury stereo data sets. This is the main contribution of the dissertation.

To learn a good matching cost, we propose training a convolutional neural network on pairs of small image patches where the true disparity is known (for example, obtained by LIDAR or structured light). The output of the network is used to initialize the matching cost. We proceed with a number of post-processing steps that are not novel, but are necessary to achieve good results. Matching costs are combined between neighboring pixels with similar image intensities using cross-based cost aggregation. Smoothness constraints are enforced by semiglobal matching, and a left-right consistency check is used to detect and eliminate errors in occluded regions. We perform subpixel enhancement and apply a median filter and a bilateral filter to obtain the final disparity map.

3.1 Matching Cost

We approach the problem of computing the matching cost by using a supervised learning approach. In this section we describe how the data set is constructed, which models are used to learn the mapping from image patches to matching costs, and how we trained them. We use `typewriter` font for the names of hyperparameters.

3.1.1 Constructing the Data Set

We use ground truth disparity maps from either the KITTI or Middlebury stereo data sets to construct a binary classification data set. At each image position where the true disparity is known we extract one negative and one positive training example. This ensures that the data set contains an equal number of positive and negative examples. A positive example is a pair of patches, one from the left and one from the right image, whose center pixels are the images of the same 3D point, while a negative example is a pair of patches where this is not the case. The following section describes the data set construction step in detail.

Let $< P^L_{n\times n}(\mathbf{p}), P^R_{n\times n}(\mathbf{q}) >$ denote a pair of patches, where $P^L_{n\times n}(\mathbf{p})$ is an $n\times n$ patch from the left image centered at position $\mathbf{p} = (x, y)$, $P^R_{n\times n}(\mathbf{q})$ is an $n \times n$ patch from the right image centered at position $\mathbf{q}$, and d denotes the correct disparity at position

p. A negative example is obtained by setting the center of the right patch to

$$\mathbf{q} = (x - d + o_{\text{neg}}, y),$$

where o_{neg} is chosen from either the interval [`dataset_neg_low`, `dataset_neg_high`] or, its origin reflected counterpart, [−`dataset_neg_high`, −`dataset_neg_low`]. The random offset o_{neg} ensures that the resulting image patches are not centered around the same 3D point.

A positive example is derived by setting

$$\mathbf{q} = (x - d + o_{\text{pos}}, y),$$

where o_{pos} is chosen randomly from the interval [−`dataset_pos`, `dataset_pos`]. The reason for including o_{pos}, instead of setting it to zero, has to do with the stereo method used later on. In particular, we found that cross-based cost aggregation performs better when the network assigns low matching costs to good matches as well as near matches. In our experiments, the hyperparameter `dataset_pos` was never larger than one pixel.

3.1.2 Network Architectures

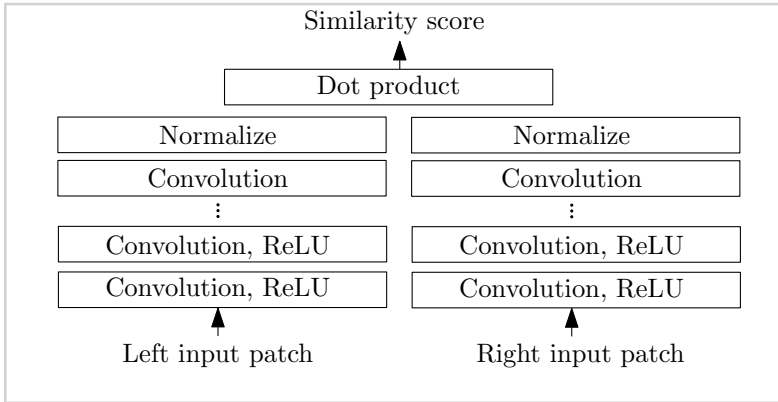
We describe two network architectures for learning a similarity measure on image patches. The first architecture is faster than the second, but produces disparity maps that are slightly less accurate. In both cases, the input to the network is a pair of small image patches and the output is a measure of similarity between them. Both architectures contain a trainable feature extractor that represents each image patch with a feature vector. The similarity between patches is measured on the feature vectors instead of the raw image intensity values. The fast architecture uses a fixed similarity measure to compare the two feature vectors, while the accurate architecture attempts to learn a good similarity measure on feature vectors.

Fast Architecture

The first architecture is a siamese network, that is, two shared-weight sub-networks joined at the head [20]. The sub-networks are composed of a number of convolutional layers with rectified linear units following all but the last layer. Both sub-networks output a vector capturing the properties of the input patch. The resulting two vectors are compared using the cosine similarity measure to produce the final output of the network. Figure 3.1 provides an overview of the architecture.

Figure 3.1

The fast architecture is a siamese network. The two sub-networks consist of a number of convolutional layers followed by rectified linear units (abbreviated "ReLU"). The similarity score is obtained by extracting a vector from each of the two input patches and computing the cosine similarity between them. On this diagram, as in our implementation, the computation of the cosine similarity is split in two steps: normalization and dot product. This way, the normalization needs to be performed only once per position (see Section 3.1.3).



The network is trained by minimizing a hinge loss. The loss is computed by considering pairs of examples centered around the same image position where one example belongs to the positive and one to the negative class. Let s_+ be the output of the network for the positive example, s_- be the output of the network for the negative example, and let m, the margin, be a positive real number. The hinge loss for that pair of examples is defined as $\max(0, m + s_- - s_+)$. The loss is zero when the similarity of the positive example is greater than the similarity of the negative example by at least the margin m. The margin was treated as a hyperparameter and was set to 0.2 after performing a grid search over the range from 0 to 1.

Since the hinge loss function depends on both the score of the positive as well as the score of the negative example, we have to ensure that both examples appear in the same mini-batch during training. If the size of the mini-batch is n, this is achieved by selecting $n/2$ random training images and positions; one positive and one negative example is extracted from each position as described in Section 3.1.1. Once the outputs of all examples of the mini-batch have been computed, the gradients are obtained by the backpropagation algorithm and the weights are updated accordingly.

The hyperparameters of this architecture are the number of convolutional layers in each sub-network (`num_conv_layers`), the size of the convolution kernels (`conv_kernel_size`), the number of feature maps in each layer (`num_conv_feature_maps`),

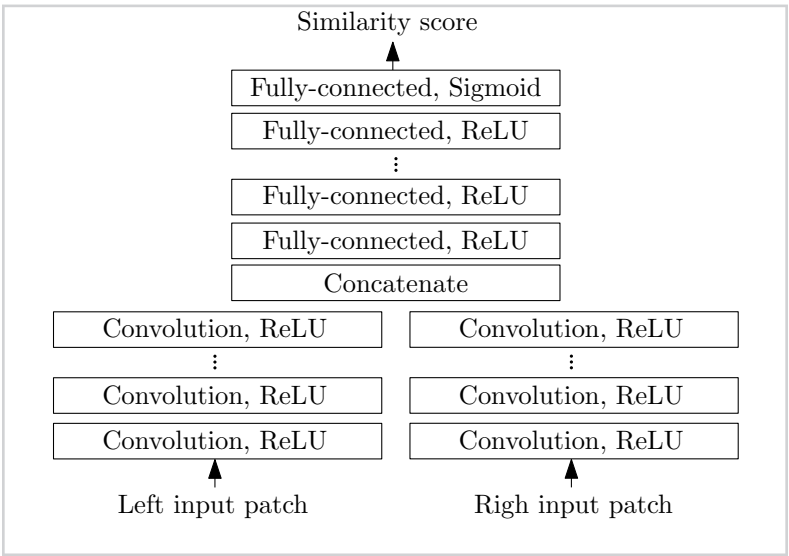


Figure 3.2

The accurate architecture begins with two convolutional feature extractors. The extracted feature vectors are then concatenated and compared by a number of fully-connected layers. The inputs are two image patches and the output is a single real number between 0 and 1, which we interpret as a measure of similarity between the input images.

and the size of the input patch (`input_patch_size`).

Accurate Architecture

The second architecture is derived from the first by replacing the cosine similarity with a number of fully-connected layers (see Figure 3.2). This architectural change increased the running time, but decreased the error rate. The two sub-networks comprise a number of convolutional layers, with a rectified linear unit following each layer. The resulting two vectors are concatenated and forward-propagated through a number of fully-connected layers followed by rectified linear units. The last fully-connected layer produces a single number which, after being transformed with the sigmoid nonlinearity, is interpreted as the similarity score between the input patches.

We use the binary cross-entropy loss for training. Let s denote the output of the network for one training example and t denote the class of that training example; $t = 1$ if the example belongs to the positive class and $t = 0$ if the example belongs to the negative class. The binary cross-entropy loss for that example is defined as $t \log(s) + (1 - t) \log(1 - s)$.

The decision to have two different loss functions, one for each architecture, was based on empirical evidence. We would have preferred to use the same loss function for both architectures, but experiments showed that the binary cross-entropy loss performed better than the hinge loss on the accurate architecture. On the other hand, since the last step of the fast architecture is the cosine similarity computation, a cross-entropy loss was not directly applicable. We observed that other researches used similar loss functions for siamese architectures (Chopra et al. [28], Hadsell et al. [55], Weston et al. [145]) and typically used a cross-entropy loss when the network resembled our accurate architecture.

The hyperparameters for the accurate architecture are the number of convolutional layers in each sub-network (`num_conv_layers`), the number of feature maps in each layer (`num_conv_feature_maps`), the size of the convolution kernels (`conv_kernel_size`), the size of the input patch (`input_patch_size`), the number of units in each fully-connected layer (`num_fc_units`), and the number of fully-connected layers (`num_fc_layers`).

Designing the Network Architecture

When developing algorithms for computing the stereo matching cost we were guided by the following two questions:

- *Should the similarity measure be computed on raw image intensity values or on feature vectors extracted from patches?*

  A possible advantage of computing the similarity on feature vectors is that the representation can learn to be insensitive to some transformations, for example small changes in brightness, viewpoint, and vertical disparity; while being sensitive to others, for example, small changes in horizontal disparity.

- *Can we design an architecture with a fast matching step?*

  The process of computing the similarity of image patches decomposes into two steps: the feature extraction step and the comparison step. In the accurate architecture, for example, feature vectors are extracted by the convolutional sub-networks and compared by the fully-connected neural network. The feature extraction step is run once, but the comparison step needs to be run once for each disparity under consideration. It is, therefore, important that the operation performed in the comparison step are efficient. This was the motivation

for the fast architecture, where the dot product is the only operation performed in the comparison step.

We were driven by empirical evidence when designing the architecture of the network. We tried several matching cost network architectures, most of which aren't described in this thesis. When experimenting with a new architecture, we computed its validation error and compared it to our baseline model. If the validation error, which was obtained using a single 80/20 split of the data set, looked promising, the model was explored further, otherwise a new architecture was tested.

We experimented with the following approaches:

- Adding max or average pooling and subsampling.
- Adding dropout [127].
- Using a hierarchical approach, that is, predicting the disparity of downsampled versions of the images and combining the results.
- Posing the stereo problem as a regression problem. Instead of performing binary classification, we tried predicting the amount of shift that would make the two patches well registered.
- Untying the weights of the sub-networks, which would make the feature extractor for the left patch compute a different function than the feature extractor for the right patch.
- Back-propagating through the linear search for the best correspondence.
- Back-propagating through semiglobal matching.
- We experimented with two alternatives to concatenating the feature vectors in the accurate architecture. We tried adding them,

$$f_i = f_i^L + f_i^R, \tag{3.1}$$

where f^L and f^R denote the feature vectors extracted from the left and right patches. We also experimented with using a bilinear projection,

$$f_i = b_i + \sum_{j,k} f_j^L W_{ijk} f_k^R, \tag{3.2}$$

where $\mathbf{W}$ is a tensor of weights and $\mathbf{b}$ is a vector of biases. A bilinear projection was used in the paper describing deep tensor neural networks by Yu et al. [151].

- Concatenating the two patches before presenting them to the network, that is, an extreme version of the accurate architecture without the sub-networks but with a single column of convolutional layers followed by fully-connected layers.

- As is evident from Table 4.12, we also experimented with many different hyper-parameters of the two chosen architectures. For example, the number of convolutional layer, the number of feature maps, and the number of fully-connected layers.

3.1.3 *Computing the Matching Cost*

The output of the network is used to initialize the matching cost:

$$C_{\text{CNN}}(\mathbf{p}, d) = -s(< P^L(\mathbf{p}), P^R(\mathbf{p} - \mathbf{d}) >),$$

where $s(< P^L(\mathbf{p}), P^R(\mathbf{p}-\mathbf{d}) >)$ is the output of the network when run on input patches $P^L(\mathbf{p})$ and $P^R(\mathbf{p}-\mathbf{d})$. The minus sign converts the similarity score to a matching cost.

To compute the entire matching cost tensor $C_{\text{CNN}}(\mathbf{p}, d)$ we would, naively, have to perform the forward pass for each image location and each disparity under consideration. The following three implementation details kept the running time manageable:

- The outputs of the two sub-networks need to be computed only once per location, and do not need to be recomputed for every disparity under consideration.

- The output of the two sub-networks can be computed for all pixels in a single forward pass by propagating full-resolution images, instead of small image patches. Performing a single forward pass on the entire $w \times h$ image is faster than performing $w \cdot h$ forward passes on small patches because many intermediate results can be reused.

  To make this step more concrete and easier to understand, let us calculate the sizes of the input and output tensors of the first convolutional layer. Assume that the batch size is 128, the number of feature maps is 112, the size of the kernel is 3×3, the patch size is 9×9, and the size of the test image is 800×600. During training the input to the first convolutional layer is $128 \times 1 \times 9 \times 9$ and

the output is $128 \times 112 \times 7 \times 7$. The spatial resolution decreases from 9×9 to 7×7 because of the boundary effects of convolution. Since the output of the convolutional layer is computed by replicating the 112 feature detectors at each position in the image, we can apply the same convolutional layer to inputs of different spatial dimensions. At test time the input to the first convolutional layer is $2 \times 1 \times 800 \times 600$ and the output is $2 \times 112 \times 800 \times 600$. The first dimension is 2, because we forward propagate the left and right image. Note that in order to preserve the spatial resolution of 800×600 we add zero padding at test time.

- The output of the fully-connected layers in the accurate architecture can also be computed in a single forward pass. This is done by replacing each fully-connected layer with a convolutional layer with 1×1 kernels. We still need to perform the forward pass for each disparity under consideration; the maximum disparity d is 228 for the KITTI data set and 400 for the Middlebury data set. As a result, the fully-connected part of the network needs to be run d times, and is a bottleneck for the accurate architecture.

To compute the matching cost of a pair of images, we run the sub-networks once on each image and run the fully-connected layers d times, where d is the maximum disparity under consideration. This insight was important in designing the architecture of the network. We could have chosen an architecture where the two images are concatenated before being presented to the network, but that would imply a large cost at runtime because the whole network would need to be run d times. This insight also led us to the fast architecture, where the only layer that is run d times is the dot product of the feature vectors.

3.2 Stereo Method

The raw outputs of the convolutional neural network are not enough to produce accurate disparity maps, with errors particularly apparent in low-texture regions and occluded areas. The quality of the disparity maps can be improved by applying a series of post-processing steps referred to as the stereo method. The stereo method we used was influenced by Mei et al. [97] and comprises cross-based cost aggregation, semiglobal matching, a left-right consistency check, subpixel enhancement, a median, and a bilateral filter.

3.2.1 Cross-based Cost Aggregation

Information from neighboring pixels can be combined by averaging the matching cost over a fixed window. This approach fails near depth discontinuities, where the assumption of constant depth within a window is violated. We might prefer a method that adaptively selects the neighborhood for each pixel, so that support is collected only from pixels of the same physical object. In cross-based cost aggregation [157] we build a local neighborhood around each location comprising pixels with similar image intensity values with the hope that these pixels belong to the same object.

The method begins by constructing an upright cross at each position; this cross is used to define the local support region. The left arm $\mathbf{p}_l$ at position $\mathbf{p}$ extends left as long as the following two conditions hold:

- $|I(\mathbf{p}) - I(\mathbf{p}_l)| <$ `cbca_intensity`; the image intensities at positions $\mathbf{p}$ and $\mathbf{p}_l$ should be similar, their difference should be less than `cbca_intensity`.
- $\|\mathbf{p} - \mathbf{p}_l\| <$ `cbca_distance`; the horizontal distance (or vertical distance in case of top and bottom arms) between positions $\mathbf{p}$ and $\mathbf{p}_l$ is less than `cbca_distance` pixels.

The right, bottom, and top arms are constructed analogously. Once the four arms are known, we can compute the support region $U(\mathbf{p})$ as the union of horizontal arms of all positions $\mathbf{q}$ laying on $\mathbf{p}$'s vertical arm (see Figure 3.3).

Zhang et al. [157] suggest that aggregation should consider the support regions of both images in a stereo pair. Let U^L and U^R denote the support regions in the left and right image. We define the combined support region U_d as

$$U_d(\mathbf{p}) = \{\mathbf{q} | \mathbf{q} \in U^L(\mathbf{p}), \mathbf{q} - \mathbf{d} \in U^R(\mathbf{p} - \mathbf{d})\}.$$

The matching cost is averaged over the combined support region:

$$\begin{aligned} C^0_{\text{CBCA}}(\mathbf{p}, d) &= C_{\text{CNN}}(\mathbf{p}, d), \\ C^i_{\text{CBCA}}(\mathbf{p}, d) &= \frac{1}{|U_d(\mathbf{p})|} \sum_{\mathbf{q} \in U_d(\mathbf{p})} C^{i-1}_{\text{CBCA}}(\mathbf{q}, d), \end{aligned}$$

where i is the iteration number. We repeat the averaging a number of times. Since the support regions are overlapping, the results can change at each iteration. We skip cross-

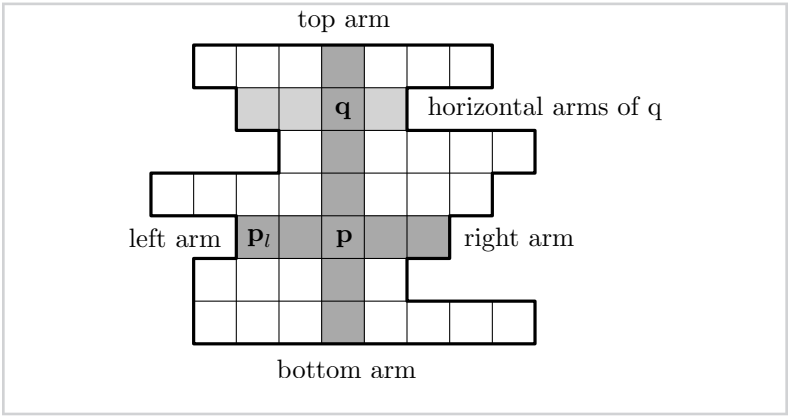


Figure 3.3

The support region for position **p** is the union of horizontal arms of all positions **q** on **p**'s vertical arm.

based cost aggregation in the fast architecture because it is not crucial for achieving a low error rate and because it is relatively expensive to compute.

3.2.2 Semiglobal Matching

We refine the matching cost by enforcing smoothness constraints on the disparity image. Following Hirschmüller [62], we define an energy function $E(D)$ that depends on the disparity image D:

$$E(D) = \sum_{\mathbf{p}} \Big(C^4_{\text{CBCA}}(\mathbf{p}, D(\mathbf{p})) + \sum_{\mathbf{q} \in N_\mathbf{p}} P_1 \cdot 1\{|D(\mathbf{p}) - D(\mathbf{q})| = 1\} + \sum_{\mathbf{q} \in N_\mathbf{p}} P_2 \cdot 1\{|D(\mathbf{p}) - D(\mathbf{q})| > 1\} \Big),$$

where $1\{\cdot\}$ denotes the indicator function. The first term penalizes disparities with high matching costs. The second term adds a penalty P_1 when the disparity of neighboring pixels differ by one. The third term adds a larger penalty P_2 when the neighboring disparities differ by more than one.

Rather than minimizing $E(D)$ in all directions simultaneously, we could perform the minimization in a single direction with dynamic programming. This solution would

introduce unwanted streaking effects, since there would be no incentive to make the disparity image smooth in the directions we are not optimizing over. In semiglobal matching we minimize the energy in a single direction, repeat for several directions, and average to obtain the final result. Although Hirschmüller [62] suggested choosing sixteen directions, we only optimized along the two horizontal and the two vertical directions; adding the diagonal directions did not improve the accuracy of our system. To minimize $E(D)$ in direction $\mathbf{r}$, we define a matching cost $C_{\mathbf{r}}(\mathbf{p}, d)$ with the following recurrence relation:

$$\begin{aligned} C_{\mathbf{r}}(\mathbf{p}, d) = C^4_{\text{CBCA}}(\mathbf{p}, d) - \min_k C_r(\mathbf{p} - \mathbf{r}, k) \\ + \min\Big\{ C_r(\mathbf{p} - \mathbf{r}, d), C_r(\mathbf{p} - \mathbf{r}, d-1) + P_1, \\ C_r(\mathbf{p} - \mathbf{r}, d+1) + P_1, \min_k C_{\mathbf{r}}(\mathbf{p} - \mathbf{r}, k) + P_2 \Big\}. \end{aligned}$$

The second term is subtracted to prevent values of $C_{\mathbf{r}}(\mathbf{p}, d)$ from growing too large and does not affect the optimal disparity map.

The penalty parameters P_1 and P_2 are set according to the image gradient so that jumps in disparity coincide with edges in the image. Let $D_1 = |I^L(\mathbf{p}) - I^L(\mathbf{p} - \mathbf{r})|$ and $D_2 = |I^R(\mathbf{p} - \mathbf{d}) - I^R(\mathbf{p} - \mathbf{d} - \mathbf{r})|$ be the difference in image intensity between two neighboring positions in the direction we are optimizing over. We set P_1 and P_2 according to the following rules:

$P_1 =$ `sgm_P1`,	$P_2 =$ `sgm_P2`	if $D_1 <$ `sgm_D`, $D_2 <$ `sgm_D`;
$P_1 =$ `sgm_P1/sgm_Q2`,	$P_2 =$ `sgm_P2/sgm_Q2`	if $D_1 \geq$ `sgm_D`, $D_2 \geq$ `sgm_D`;
$P_1 =$ `sgm_P1/sgm_Q1`,	$P_2 =$ `sgm_P2/sgm_Q1`	otherwise.

The hyperparameters `sgm_P1` and `sgm_P2` set a base penalty for discontinuities in the disparity map. The base penalty is reduced by a factor of `sgm_Q1` if one of D_1 or D_2 indicate a strong image gradient or by a larger factor of `sgm_Q2` if both D_1 and D_2 indicate a strong image gradient. The value of P_1 is further reduced by a factor of `sgm_V` when considering the two vertical directions; in the ground truth, small changes in disparity are much more frequent in the vertical directions than in the horizontal directions and should be penalised less.

The final cost $C_{\text{SGM}}(\mathbf{p}, d)$ is computed by taking the average across all four directions:

$$C_{\text{SGM}}(\mathbf{p}, d) = \frac{1}{4} \sum_{\mathbf{r}} C_{\mathbf{r}}(\mathbf{p}, d).$$

After semiglobal matching we repeat cross-based cost aggregation, as described in the previous section. Hyperparameters `cbca_num_iterations_1` and `cbca_num_iterations_2` determine the number of cross-based cost aggregation iterations before and after semiglobal matching.

3.2.3 Computing the Disparity Image

The disparity image $D(\mathbf{p})$ is computed by the winner-take-all strategy, that is, by finding the disparity d that minimizes $C(\mathbf{p}, d)$:

$$D(\mathbf{p}) = \operatorname*{argmin}_{d} C(\mathbf{p}, d).$$

Interpolation

The interpolation steps attempt to resolve conflicts between the disparity map predicted for the left image and the disparity map predicted for the right image. Let D^L denote the disparity map obtained by treating the left image as the reference image—this was the case so far, that is, $D^L(\mathbf{p}) = D(\mathbf{p})$—and let D^R denote the disparity map obtained by treating the right image as the reference image. D^L and D^R sometimes disagree on what the correct disparity at a particular position should be. We detect these conflicts by performing a left-right consistency check. We label each position $\mathbf{p}$ by applying the following rules in turn:

correct	if $\lvert d - D^R(\mathbf{p} - \mathbf{d})\rvert \leq 1$ for $d = D^L(\mathbf{p})$,
mismatch	if $\lvert d - D^R(\mathbf{p} - \mathbf{d})\rvert \leq 1$ for any other d,
occlusion	otherwise.

For positions marked as *occlusion*, we want the new disparity value to come from the background. We interpolate by moving left until we find a position labeled *correct* and use its value. For positions marked as *mismatch*, we find the nearest *correct* pixels in 16 different directions and use the median of their disparities for interpolation. We refer to the interpolated disparity map as D_{INT}.

Subpixel Enhancement

Subpixel enhancement provides an easy way to increase the resolution of a stereo algorithm. We fit a quadratic curve through the neighboring costs to obtain a new disparity image:

$$D_{\text{SE}}(\mathbf{p}) = d - \frac{C_+ - C_-}{2(C_+ - 2C + C_-)},$$

where $d = D_{\text{INT}}(\mathbf{p})$, $C_- = C_{\text{SGM}}(\mathbf{p}, d-1)$, $C = C_{\text{SGM}}(\mathbf{p}, d)$, and $C_+ = C_{\text{SGM}}(\mathbf{p}, d+1)$.

Refinement

The final steps of the stereo method consist of a 5×5 median filter and the following bilateral filter:

$$D_{\text{BF}}(\mathbf{p}) = \frac{1}{W(\mathbf{p})} \sum_{\mathbf{q} \in N_\mathbf{p}} D_{\text{SE}}(\mathbf{q}) \cdot g(\|\mathbf{p} - \mathbf{q}\|) \cdot 1\{|I^L(\mathbf{p}) - I^L(\mathbf{q})| < \texttt{blur\_threshold}\},$$

where $g(x)$ is the probability density function of a zero mean normal distribution with standard deviation `blur_sigma` and $W(\mathbf{p})$ is the normalizing constant:

$$W(\mathbf{p}) = \sum_{\mathbf{q} \in N_\mathbf{p}} g(\|\mathbf{p} - \mathbf{q}\|) \cdot 1\{|I^L(\mathbf{p}) - I^L(\mathbf{q})| < \texttt{blur\_threshold}\}.$$

The role of the bilateral filter is to smooth the disparity map without blurring the edges. D_{BF} is the final output of our stereo method.

4

Evaluation

We used three stereo data sets in our experiments: KITTI 2012, KITTI 2015, and Middlebury. The test set error rates reported in Tables 4.1, 4.2, and 4.4 were obtained by submitting the generated disparity maps to the online evaluation servers, while the validation error rates were computing using hold-out cross validation, with 80 % of the examples used for training and 20 % for testing.

4.1 KITTI Stereo Data Set

The KITTI stereo data set [49, 98] is a collection of rectified image pairs taken from two video cameras mounted on the roof of a car, roughly 54 centimeters apart. The images were recorded while driving in and around the city of Karlsruhe, in sunny and cloudy weather, at daytime. The images were taken at a resolution of 1240×376. A rotating laser scanner mounted behind the left camera recorded ground truth depth, labeling around 30 % of the image pixels.

The ground truth disparities for the test set are withheld and an online leaderboard is provided where researchers can evaluate their method on the test set. Submissions are allowed once every three days. Error is measured as the percentage of pixels where the true disparity and the predicted disparity differ by more than three pixels. Translated into distance, this means that, for example, the error tolerance is 3 centimeters for objects 2 meters from the camera and 80 centimeters for objects 10 meters from the camera.

Two KITTI stereo data sets exist: KITTI 2012[1] and, the newer, KITTI 2015[2]. For the task of computing stereo they are nearly identical, with the newer data set improving some aspects of the optical flow task. The 2012 data set contains 194 training and 195 testing images, while the 2015 data set contains 200 training and 200 testing images. There is a subtle but important difference introduced in the newer data set: vehicles in motion are densely labeled and car glass is included in the evaluation. This emphasizes the method's performance on reflective surfaces. A subset of the training set images, together with ground truth disparity maps, is shown in Figure A.1 for the KITTI 2012 data set and in Figure A.4 for the KITTI 2015 data set.

The best performing methods on the KITTI 2012 data set are listed in Table 4.1.

[1] The KITTI 2012 scoreboard: http://www.cvlibs.net/datasets/kitti/eval_stereo_flow.php?benchmark=stereo

[2] The KITTI 2015 scoreboard: http://www.cvlibs.net/datasets/kitti/eval_scene_flow.php?benchmark=stereo

Table 4.1

The highest ranking methods on the KITTI 2012 data set as of October 2015. The "Setting" column provides insight into how the disparity map is computed: "F" indicates the use of optical flow, "MV" indicates more than two temporally adjacent images, and "MS" indicates the use of epipolar geometry for computing the optical flow. The "Error" column reports the percentage of misclassified pixels and the "Runtime" column measures the time, in seconds, required to process one pair of images.

Rank	Method		Setting	Error	Runtime
1	*MC-CNN-acrt*	*Accurate architecture*		2.43	67
2	Displets	Güney and Geiger [54]		2.47	265
3	MC-CNN	Žbontar and LeCun [154]		2.61	100
4	PRSM	Vogel et al. [142]	F, MV	2.78	300
	MC-CNN-fst	*Fast architecture*		2.82	0.8
5	SPS-StFl	Yamaguchi et al. [148]	F, MS	2.83	35
6	VC-SF	Vogel et al. [141]	F, MV	3.05	300
7	Deep Embed	Chen et al. [26]		3.10	3
8	JSOSM	Unpublished work		3.15	105
9	OSF	Menze and Geiger [98]	F	3.28	3000
10	CoR	Chakrabarti et al. [25]		3.30	6

Our accurate architecture ranks first with an error rate of 2.43 %. Third place on the leaderboard is held by our previous work [154] with an error rate of 2.61 %. The two changes that reduced the error from 2.61 % to 2.43 % were augmenting the data set (see Section 4.5) and doubling the number of convolution layers while reducing the kernel size from 5×5 to 3×3. The method in second place [54] uses the matching cost computed by our previous work [154]. The test error rate of the fast architecture is 2.82 %, which would be enough for fifth place had the method been allowed to appear in the public leaderboard. The running time for processing a single image pair is 67 seconds for the accurate architecture and 0.8 seconds for the fast architecture. Figure 4.1 contains a pair of examples from the KITTI 2012 data set, together with the predictions of our method.

Table 4.2 presents the frontrunners on the KITTI 2015 data sets. The error rates of our methods are 3.89 % for the accurate architecture and 4.46 % for the fast architecture, occupying first and second place on the leaderboard. Since one submission

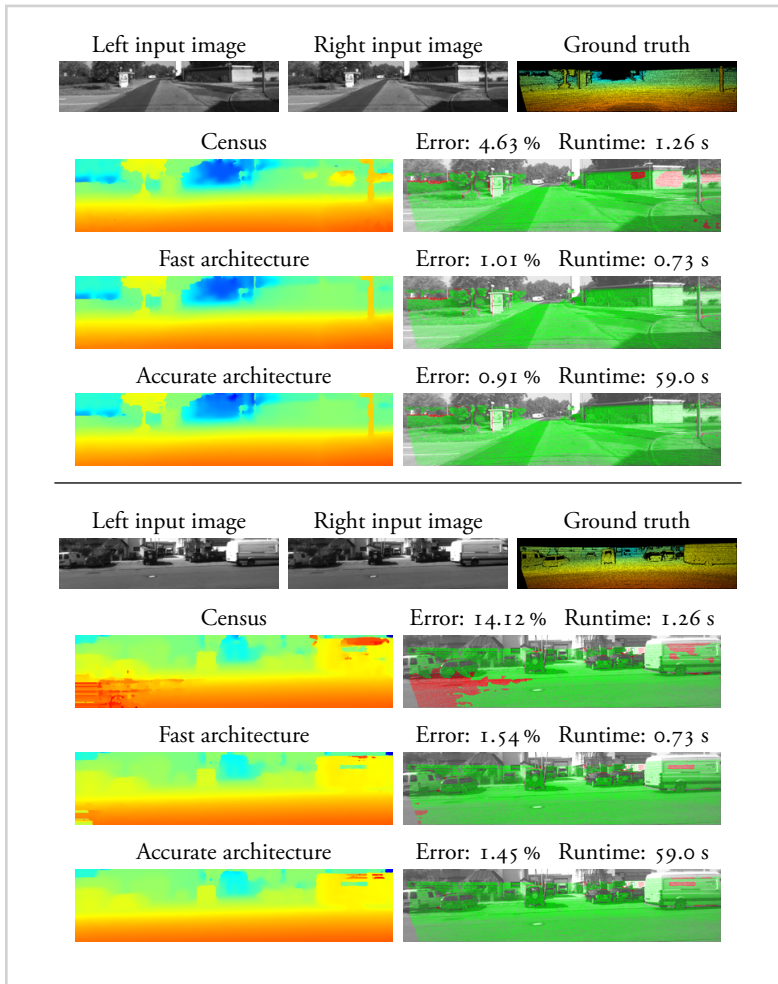


Figure 4.1

Examples of predicted disparity maps on the KITTI 2012 data set. Note how some regions of the image (the white wall in the top example, and the asphalt in the bottom example) cause problems for the census transform. The fast and the accurate architecture perform better, with the accurate architecture making fewer mistakes on average. When comparing running times note that our implementation of the Census transform was not optimized for speed.

Table 4.2

The leading submission on the KITTI 2015 leaderboard as of October 2015. The "Setting", "Error", and "Runtime" columns have the same meaning as in Table 4.1.

Rank	Method		Setting	Error	Runtime
1	*MC-CNN-acrt*	*Accurate architecture*		3.89	67
	MC-CNN-fst	*Fast architecture*		4.62	0.8
2	SPS-St	Yamaguchi et al. [148]		5.31	2
3	OSF	Menze and Geiger [98]	F	5.79	3000
4	PR-Sceneflow	Vogel et al. [140]	F	6.24	150
5	SGM+C+NL	Hirschmüller [62], Sun et al. [129]	F	6.84	270
6	SGM+LDOF	Brox and Malik [23], Hirschmüller [62]	F	6.84	86
7	SGM+SF	Hirschmüller [62], Hornacek et al. [67]	F	6.84	2700
8	ELAS	Geiger et al. [48]		9.72	0.3
9	OCV-SGBM	Hirschmüller [62]		10.86	1.1
10	SDM	Kostková and Sára [82]		11.96	60

per paper is allowed, only the result of the accurate architecture appears on the public leaderboard. See Figure 4.2 for the disparity maps produced by our method on the KITTI 2015 data set.

4.2 Middlebury Stereo Data Set

The image pairs of the Middlebury stereo data set are indoor scenes taken under controlled lighting conditions. Structured light was used to measure the true disparities with higher density and precision than in the KITTI data set. The data sets were published in five separates works in the years 2001, 2003, 2005, 2006, and 2014 [63, 111, 113–115]. In this paper, we refer to the Middlebury data set as the concatenation of all five data sets; a summary of each is presented in Table 4.3. A subset of the training set images, together with ground truth disparity maps, is shown in Figure A.7.

Each scene in the 2005, 2006, and 2014 data set was taken under a number of lighting conditions and shutter exposures, with a typical image pair taken under four lighting conditions and seven exposure settings for a total of 28 images of the same

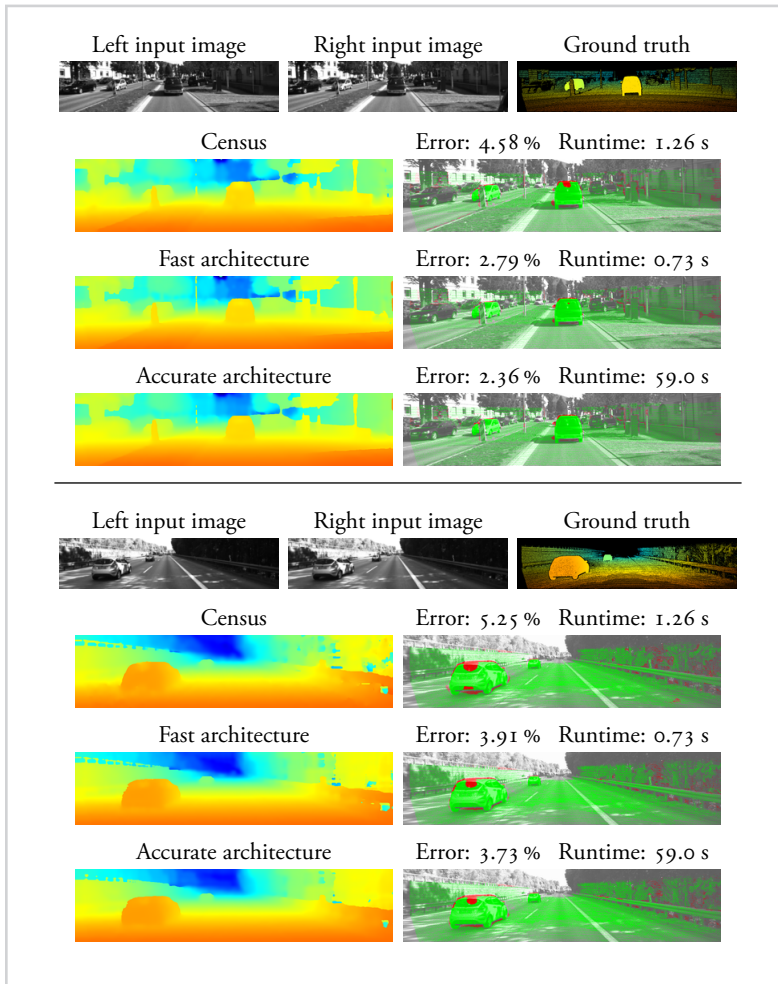


Figure 4.2

Examples of predictions on the KITTI 2015 data set. Observe that vehicles in motion are labeled densely in the KITTI 2015 data set. When comparing running times note that our implementation of the Census transform was not optimized for speed.

Table 4.3

A summary of the five Middlebury stereo data sets. The column "Number of Image Pairs" counts only the image pairs for which ground truth is available. The 2005 and 2014 data sets additionally contain a number of image pairs with ground truth disparities withheld; these image pairs constitute the test set.

Year	Number of Image Pairs	Resolution	Maximum Disparity
2001	8	380×430	30
2003	2	1800×1500	220
2005	6	1400×1100	230
2006	21	1400×1100	230
2014	23	3000×2000	800

scene.

An online leaderboard[3], similar to the one provided by KITTI, displays a ranked list of all submitted methods. Participants have only one opportunity to submit their results on the test set to the public leaderboard. This rule is stricter than the one on the KITTI data set, where submissions are allowed every three days. The test set contains 15 images borrowed from the 2005 and 2014 data sets.

The data set is provided in full, half, and quarter resolution. The error is computed at full resolution; if the method outputs half or quarter resolution disparity maps, they are upsampled before the error is computed. We chose to run our method on half resolution images because of the limited size of the graphic card's memory available.

Rectifying a pair of images using standard calibration procedures, like the ones present in the OpenCV library, results in vertical disparity errors of up to nine pixels on the Middlebury data set [115]. Each stereo pair in the 2014 data set is rectified twice: once using a standard, imperfect approach, and once using precise 2D correspondences for perfect rectification [115]. We train the network on imperfectly rectified image pairs, since only two of the fifteen test images (*Australia* and *Crusade*) are rectified perfectly.

The error is measured as the percentage of pixels where the true disparity and the predicted disparity differ by more than two pixels; this corresponds to an error tolerance of one pixel at half resolution. The error on the evaluation server is, by default,

[3]The Middlebury scoreboard: http://vision.middlebury.edu/stereo/eval3/

Table 4.4

The top ten methods on the Middlebury stereo data set as of October 2015. The "Error" column is the weighted average error after upsampling to full resolution and "Runtime" is the time, in seconds, required to process one pair of images.

Rank	Method		Resolution	Error	Runtime
1	*MC-CNN-acrt*	*Accurate architecture*	Half	8.29	150
2	MeshStereo	Zhang et al. [156]	Half	13.4	65.3
3	LCU	Unpublished work	Quarter	17.0	6567
4	TMAP	Psota et al. [105]	Half	17.1	2435
5	IDR	Kowalczuk et al. [83]	Half	18.4	0.49
6	SGM	Hirschmüller [62]	Half	18.7	9.90
7	LPS	Sinha et al. [124]	Half	19.4	9.52
8	LPS	Sinha et al. [124]	Full	20.3	25.8
9	SGM	Hirschmüller [62]	Quarter	21.2	1.48
10	SNCC	Einecke and Eggert [39]	Half	22.2	1.38

computed only on non-occluded pixels. The final error reported online is the weighted average over the fifteen test images, with the weights set by the authors of the data set.

Table 4.4 contains a snapshot of the third, and newest, version of the Middlebury leaderboard. Our method ranks first with an error rate of 8.29 % and a substantial lead over the second placed MeshStereo method, whose error rate is 13.4 %. See Figure 4.3 for disparity maps produced by our method on one image pair from the Middlebury data set.

4.3 Statistical Significance

We use the procedure recommended by Demšar [35] to determine whether the performance of the accurate architecture is significantly better than the performance of existing approaches. The statistical test operates on a $k \times N$ matrix of scores, where k is the number of algorithms and N is the number of data sets. The error rates are converted into ranks and the average rank for each method is computed. The Friedman test is used in an attempt to reject the null hypothesis, which states that the algorithms are equivalent and, therefore, have the same average rank. If the null hypothesis is rejected we can proceed with a posthoc analysis. Since we are comparing a single method

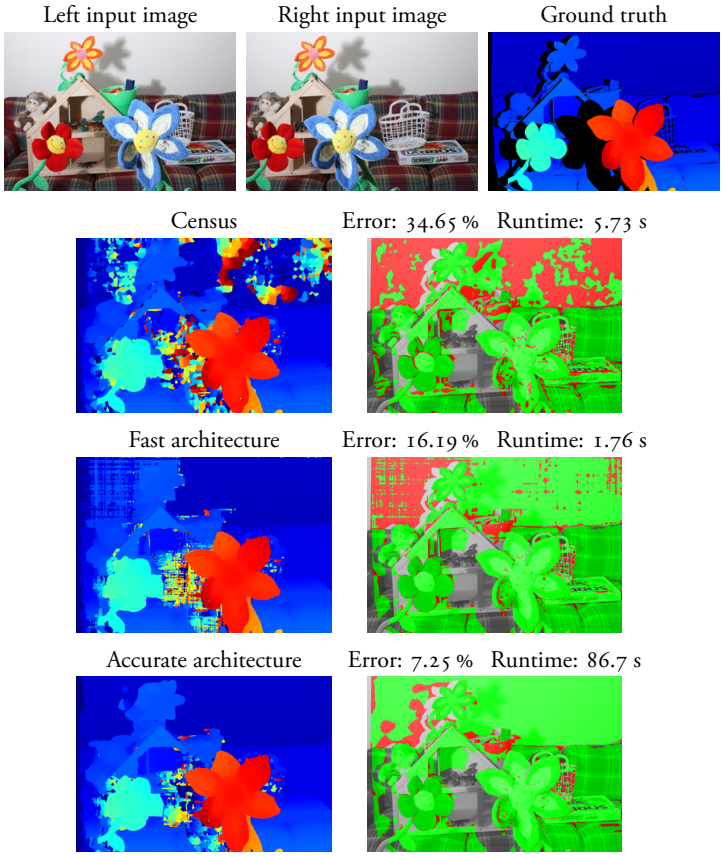


Figure 4.3

An example of a particularly difficult image pair from the Middlebury data set; the white wall in the background is practically textureless. The accurate architecture is able to classify most of it correctly. The fast architecture doesn't do as well but still performs better than census. When comparing running times note that our implementation of the Census transform was not optimized for speed.

to existing approaches we use the Bonferroni-Dunn posthoc test. The critical difference is computed at $\alpha = 0.05$ (see Demšar [35] for details). If the average rank of the proposed method is smaller than the average rank of a competing method by at least the critical difference, we can conclude that the proposed method is significantly better.

We would, ideally, compute the error rates of all methods on all datasets, but since the source code of most methods is not made available, we are limited to the information provided by the KITTI and Middlebury websites. Since modern stereo methods are complicated programs, composed of several complex subroutines, reimplementing the top performing methods from the description in the paper is not always possible. Because most authors evaluated their methods on only one of the three stereo data sets, an analysis across datasets is not possible. Nevertheless, a statistical analysis is possible within each data set, that is, a separate analysis for KITTI 2012, KITTI 2015, and Middlebury. We perform the analysis by treating one stereo test pair as a separate dataset. We justify this decision by noting that one stereo pair contains many test points (up to 6 million on the Middlebury data set) and therefore provides a reliable estimate of the algorithms' performance. A downside of the analysis is that we were only able to compare the accurate architecture, since we were not able to obtain the test error rates for the fast architecture due to the limited number of submissions. The results of the statistical analysis are reported in Table 4.5.

The KITTI website displays the error rates on 20 out of the 200 test stereo pairs for each method. The average ranks are computed based on this information for the 10 top-performing methods. On the KITTI 2012 data set our method achieves an average rank of 3.40. The critical difference is 2.65 and we can conclude that MC-CNN-acrt significantly outperforms the following methods: Deep Embed, JSOSM, OSF, and CoR. On the KITTI 2015 dataset MC-CNN-acrt achieves an average rank of 1.05. The critical difference is, again, 2.65 and we can conclude that it significantly outperforms all but the SPS-St method.

The Middlebury website contains the error rates on all 15 stereo test pairs. However, since some test pairs contain the same scene with different illumination, the error rates are not independent. We remove all similar stereo pairs and only use the following 10 test pairs: *Austr*, *Bicyc2*, *Class*, *Compu*, *Crusa*, *Djemb*, *Hoops*, *Livgrm*, *Nkuba*, *Plants*, and *Stairs*. On the Middlebury dataset our method achieves an average rank of 1.00 and, with the critical difference being 3.58, is significantly better than all but the

Table 4.5

The average ranks of the top performing stereo methods on the KITTI 2012, KITTI 2015, and Middlebury data sets. Horizontal lines are used to indicate statistical significance; MC-CNN-acrt is significantly better than all methods that are listed below the horizontal line.

KITTI 2012		KITTI 2015		Middlebury	
Method	Avg. Rank	Method	Avg. Rank	Method	Avg. Rank
MC-CNN-acrt	3.40	*MC-CNN-acrt*	1.05	*MC-CNN-acrt*	1.00
Displets	3.40	SPS-St	2.45	MeshStereo	4.18
MC-CNN	3.85	OSF	4.50	LCU	5.27
PRSM	4.78	PR-Sceneflow	5.83	TMAP	4.41
SPS-StFl	5.10	SGM+C+NL	4.88	IDR	6.00
VC-SF	5.70	SGM+LDOF	4.88	SGM_H	6.27
Deep Embed	6.60	SGM+SF	4.88	LPS_H	4.91
JSOSM	7.05	ELAS	8.95	LPS_F	5.77
OSF	7.67	OCV-SGBM	8.30	SGM_Q	8.36
CoR	7.45	SDM	9.30	SNCC	8.82

MeshStereo method.

In this section we were able to show that the accurate architecture significantly outperforms several existing approaches on KITTI 2012, KITTI 2015, or Middlebury, but the experimental data was insufficient to reach any conclusions regarding the following methods: Displets, PRSM, SPS-StFL, VC-SF, SPS-St, and MeshStereo. We should emphasize, however, that not being able to detect a significant difference does not imply that the methods perform equally well.

4.4 Details of Learning

We construct a binary classification data set from all available image pairs in the training set. The data set contains 25 million examples on the KITTI 2012, 17 million examples on the KITTI 2015, and 38 million examples on the Middlebury data set.

At training time, the input to the network was a batch of 128 pairs of image patches. At test time, the input was the entire left and right image. We could have used entire

Table 4.6

The hyperparameter values we used for the fast and accurate architectures (abbreviated "fst" and "acrt"). Note that hyperparameters concerning image intensity values (`cbca_intensity` and `sgm_D`) apply to the preprocessed images and not to raw images with intensity values in the range from 0 to 255.

	KITTI 2012		KITTI 2015		Middlebury	
Hyperparameter	fst	acrt	fst	acrt	fst	acrt
`input_patch_size`	9×9	9×9	9×9	9×9	11×11	11×11
`num_conv_layers`	4	4	4	4	5	5
`num_conv_feature_maps`	64	112	64	112	64	112
`conv_kernel_size`	3	3	3	3	3	3
`num_fc_layers`		4		4		3
`num_fc_units`		384		384		384
`dataset_neg_low`	4	4	4	4	1.5	1.5
`dataset_neg_high`	10	10	10	10	6	18
`dataset_pos`	1	1	1	1	0.5	0.5
`cbca_intensity`		0.13		0.03		0.02
`cbca_distance`		5		5		14
`cbca_num_iterations_1`		2		2		2
`cbca_num_iterations_2`		0		4		16
`sgm_P1`	4	1.32	2.3	2.3	2.3	1.3
`sgm_P2`	223	32	42.3	55.8	55.9	18.1
`sgm_Q1`	3	3	3	3	4	4.5
`sgm_Q2`	7.5	6	6	6	8	9
`sgm_V`	1.5	2	1.25	1.75	1.5	2.75
`sgm_D`	0.02	0.08	0.08	0.08	0.08	0.13
`blur_sigma`	7.74	6	4.64	6	6	1.7
`blur_threshold`	5	6	5	5	2	2

images during training as well, as it would allow us to implement the speed optimizations described in Section 3.1.3. There were several reasons why we preferred to train on image patches: it was easier to control the batch size, the examples could be shuffled so that one batch contained patches from several different images, and it was easier to maintain the same number of positive and negative examples within a batch.

We minimized the loss using mini-batch gradient descent with the momentum term set to 0.9. We trained for 14 epochs with the learning rate initially set to 0.003 for the accurate architecture and 0.002 for the fast architecture. The learning rate was decreased by a factor of 10 on the 11^{th} epoch. The number of epochs, the initial learning rate, and the learning rate decrease schedule where treated as hyperparameters and were optimized with cross-validation. Each image was preprocessed by subtracting the mean and dividing by the standard deviation of its pixel intensity values. The left and right image of a stereo pair were preprocessed separately. Our initial experiments suggested that using color information does not improve the quality of the disparity maps; therefore, we converted all color images to grayscale.

The post-processing steps of the stereo method were implemented in CUDA [101], the network training was done with the Torch environment [32] using the convolution routines from the cuDNN library [27]. The OpenCV library [19] was used for the affine transformation in the data augmentation step.

We did not use an algorithm for hyperparameter tuning. Instead, we performed a manual search of the hyperparameter space, guided by intuition. As an alternative, several algorithms for hyperparameter optimization were published recently [10, 11, 125, 149] and can be used instead. The hyperparameters we selected are shown in Table 4.6.

4.5 Data Set Augmentation

Augmenting the data set by repeatedly transforming the training examples is a commonly employed technique to reduce the network's generalization error. The transformations are applied at training time and do not affect the runtime performance. We randomly rotate, scale and shear the training patches; we also change their brightness and contrast. Since the transformations are applied to patches after they have been extracted from the images, the data augmentation step does not alter the ground truth disparity map or ruin the rectification.

The parameters of the transformation are chosen randomly for each pair of patches,

and after one epoch of training, when the same example is being presented to the network for the second time, new random parameters are selected. We choose slightly different transformation parameters for the left and right image; for example, we would rotate the left patch by 10 degrees and the right by 14. Different data sets benefited from different types of transformations and, in some cases, using the wrong transformations increased the error.

On the Middlebury data set we took advantage of the fact that the images were taken under different lighting conditions and different shutter exposures by training on all available images. The same data set augmentation parameters were used for the KITTI 2012 and KITTI 2015 data sets.

The Middlebury test data sets contains two images worth mentioning: *Classroom*, where the right image is underexposed and, therefore, darker than the left; and *Djembe*, where the left and right images were taken under different light conditions. To handle these two cases we train, 20 % of the time, on images where either the shutter exposure or the arrangements of lights are different for the left and right image.

We combat imperfect rectification on the Middlebury data set by including a small vertical disparity between the left and right image patches.

Before describing the steps of data augmentation, let us introduce some notation: in the following, a word in `typewriter` is used to denote the name of a hyperparameter defining a set, while the same word in *italic* is used to denote a number drawn randomly from that set. For example, `rotate` is a hyperparameter defining the set of possible rotations and *rotate* is a number drawn randomly from that set. The steps of data augmentation are presented in the following list:

- Rotate the left patch by *rotate* degrees and the right patch by *rotate* + *rotate_diff* degrees.
- Scale the left patch by *scale* and the right patch by *scale* · *scale_diff*.
- Scale the left patch in the horizontal direction by *horizontal_scale* and the right patch by *horizontal_scale* · *horizontal_scale_diff*.
- Shear the left patch in the horizontal direction by *horizontal_shear* and the right patch by *horizontal_shear* + *horizontal_shear_diff*.
- Translate the right patch in the vertical direction by *vertical_disparity*.

Table 4.7

The hyperparameters governing data augmentation and how they affect the validation error. The "Error" column reports the validation error when a particular data augmentation step is not used. The last two rows report validation errors with and without data augmentation. For example, the validation error on the KITTI 2012 is 2.73 % if no data augmentation is used, 2.65 % if all steps except rotation are used, and 2.61 % if all data augmentation steps are used.

	KITTI 2012		Middlebury	
Hyperparameter	Range	Error	Range	Error
`rotate`	$[-7, 7]$	2.65	$[-28, 28]$	7.99
`scale`			$[0.8, 1]$	8.17
`horizontal_scale`	$[0.9, 1]$	2.62	$[0.8, 1]$	8.08
`horizontal_shear`	$[0, 0.1]$	2.61	$[0, 0.1]$	7.91
`brightness`	$[0, 0.7]$	2.61	$[0, 1.3]$	8.16
`contrast`	$[1, 1.3]$	2.63	$[1, 1.1]$	7.95
`vertical_disparity`			$[0, 1]$	8.05
`rotate_diff`			$[-3, 3]$	8.00
`horizontal_scale_diff`			$[0.9, 1]$	7.97
`horizontal_shear_diff`			$[0, 0.3]$	8.05
`brightness_diff`	$[0, 0.3]$	2.63	$[0, 0.7]$	7.92
`contrast_diff`			$[1, 1.1]$	8.01
No data set augmentation		2.73		8.75
Full data set augmentation		2.61		7.91

- Adjust the brightness and contrast by setting the left and right image patches to:

$$P^L \leftarrow P^L \cdot \mathit{contrast} + \mathit{brightness} \text{ and}$$
$$P^R \leftarrow P^R \cdot (\mathit{contrast} \cdot \mathit{contrast\_diff}) + (\mathit{brightness} + \mathit{brightness\_diff}),$$

  with addition and multiplication carried out element-wise where appropriate.

Table 4.7 contains the hyperparameters used and measures how each data augmentation step affected the validation error.

Data augmentation reduced the validation error from 2.73 % to 2.61 % on the KITTI 2012 data set and from 8.75 % to 7.91 % on the Middlebury data set.

4.6 Runtime

We measure the runtime of our implementation on a computer with a NVIDIA Titan X graphics processor unit. Table 4.8 contains the runtime measurements across a range of hyperparameter settings for three data sets: KITTI, Middlebury half resolution, and a new, fictitious data set, called Tiny. The Tiny data set is not a real stereo dataset; it contains only one image pair comprising two black images of size 320×240. We run our method on this data set only to measure its runtime on the kind of images typically used for autonomous driving or robotics. The sizes of images we measured the runtime on were: 1242×350 with 228 disparity levels for the KITTI data set, 1500×1000 with 200 disparity levels for the Middlebury data set, and 320×240 with 32 disparity levels for the Tiny data set.

Table 4.8 reveals that the fast architecture is up to 90 times faster than the accurate architecture. Furthermore, the running times of the fast architecture are 0.78 seconds on KITTI, 2.03 seconds on Middlebury, and 0.06 seconds on the Tiny data set. We can also see that the fully-connected layers are responsible for most of the runtime in the accurate architecture, as the hyperparameters controlling the number of convolutional layer and the number of feature maps have only a small effect on the runtime.

Training times depended on the size of the data set and the architecture, but never exceeded two days.

4.7 Matching Cost

We argue that the low error rate of our method is due to the convolutional neural network and not a superior stereo method. We verify this claim by replacing the convolutional neural network with three standard approaches for computing the matching cost:

- *The sum of absolute differences* computes the matching by summing the absolute differences in image intensities between corresponding locations. See Equation 2.80. We used 9×9 patches.

- *The census transform* represents each image position as a bit vector. The size of this vector is a hyperparameter whose value, after examining several, we set to 81. The vector is computed by cropping a 9×9 image patch centered around the position of interest and comparing the intensity values of each pixel in the

Table 4.8

The time, in seconds, required to compute the matching cost, that is, the time spent in the convolutional neural network without any post-processing steps. The time does include computing the matching cost twice: once when the left image is taken to be the reference image and once when the right image is taken to be the reference image. We measure the runtime as a function of four hyperparameters controlling the network architecture; for example, the first six rows contain the runtime as the number of convolutional layers in the network increases from one to six. The last row of the table contains the running time for the entire method, including the post-processing steps. As before, we abbreviate the fast and accurate architectures as "fst" and "acrt".

		KITTI		Middlebury		Tiny	
Hyperparameter		fst	acrt	fst	acrt	fst	acrt
num_conv_layers	1	0.23	66.1	0.70	74.9	0.01	1.7
	2	0.26	66.2	0.82	75.2	0.01	1.8
	3	0.30	66.3	0.97	75.4	0.02	1.8
	4	0.34	66.4	1.11	75.6	0.03	1.8
	5	0.38	66.5	1.24	75.7	0.03	1.9
	6	0.42	66.7	1.37	76.0	0.04	1.9
num_conv_feature_maps	16	0.09	59.4	0.27	64.8	0.01	1.6
	32	0.15	60.4	0.51	66.2	0.01	1.6
	48	0.25	61.5	0.94	68.2	0.02	1.7
	64	0.34	62.7	1.24	70.0	0.03	1.7
	80	0.44	64.0	1.63	72.0	0.04	1.8
	96	0.53	65.3	1.93	73.9	0.04	1.8
	112	0.61	66.4	2.28	75.7	0.05	1.8
	128	0.71	67.7	2.61	77.8	0.06	1.9
num_fc_layers	1		16.3		25.3		0.5
	2		32.9		50.7		0.9
	3		49.6		75.7		1.4
	4		66.4		101.2		1.8
	5		82.9		126.4		2.3
num_fc_units	128		17.4		21.4		0.6
	256		38.5		44.9		1.1
	384		66.4		75.7		1.8
	512		101.0		113.3		2.7
No stereo method		0.34	66.4	1.24	75.7	0.03	1.8
Full stereo method		0.78	67.1	2.03	84.8	0.06	1.9

patch to the intensity value of the pixel in the center. When the center pixel is brighter the corresponding bit is set. See Equation 2.91. The matching cost is computed as the hamming distance between two census transformed vectors.

- *Normalized cross-correlation* was defined in Equation 2.87. It computes the same function as the last two layers of the fast architecture (normalization and dot product). The neighbourhood $N_{\mathbf{p}}$ was set to a square 11×11 window around $\mathbf{p}$.

The "sad", "cens", and "ncc" columns of Table 4.9 contain the results of the sum of absolute differences, the census transform, and normalized cross-correlation on the KITTI 2012, KITTI 2015, and Middlebury data sets. The validation errors in the last rows of Table 4.9 should be used to compare the five methods. On all three data sets the accurate architecture performs best, followed by the fast architecture, which in turn is followed by the census transform. These are the three best performing methods on all three data sets. Their error rates are 2.61 %, 3.02 %, and 4.90 % on KITTI 2012; 3.25 %, 3.99 %, and 5.03 % on KITTI 2015; and 7.91 %, 9.87 %, and 16.72 % on Middlebury. The sum of absolute differences and the normalized cross-correlation matching costs produce disparity maps with larger errors. For a visual comparison of our method and the census transform see Figures 4.1, 4.2, and 4.3.

4.8 Stereo Method

The stereo method includes a number of post-processing steps: cross-based cost aggregation, semiglobal matching, interpolation, subpixel enhancement, a median, and a bilateral filter. Disparity maps obtained by terminating the stereo method after each post-processing step are displayed in Figure 4.4.

We ran a set of experiments in which we exclude each of the aforementioned steps and recorded the validation error (see Table 4.9). The last two rows of Table 4.9 allude to the importance of the post-processing steps of the stereo method. We see that, if all post-processing steps are removed, the validation error of the accurate architecture increases from 2.61 % to 13.49 % on KITTI 2012, from 3.25 % to 13.38 % on KITTI 2015, and from 7.91 % to 28.33 % on Middlebury.

Out of all post-processing steps of the stereo method, semiglobal matching affects the validation error the strongest. If we remove it, the validation error increases from

Table 4.9

The numbers measure validation error when a particular post-processing step is excluded from the stereo method. The last two rows of the tables should be interpreted differently: they contain the validation error of the raw convolutional neural network and the validation error after the complete stereo method. For example, if we exclude semiglobal matching, the fast architecture achieves an error rate of 8.78 % on the KITTI 2012 data set and an error rate of 3.02 % after applying the full stereo method. We abbreviate the method names as "fst" for the fast architecture, "acrt" for the accurate architecture, "sad" for the sum of absolute differences, "cens" for the census transform, and "ncc" for the normalized cross-correlation matching cost.

	KITTI 2012				
	fst	acrt	sad	cens	ncc
Cross-based cost aggregation	3.02	2.73	8.22	5.21	8.93
Semiglobal matching	8.78	4.26	19.58	8.84	10.72
Interpolation	3.48	2.96	9.21	5.96	11.16
Subpixel Enhancement	3.03	2.65	8.16	4.95	8.93
Median filter	3.03	2.63	8.16	4.92	9.00
Bilateral filter	3.26	2.79	8.75	5.70	9.76
No stereo method	15.70	13.49	32.30	53.55	22.21
Full stereo method	3.02	2.61	8.16	4.90	8.93
	KITTI 2015				
	fst	acrt	sad	cens	ncc
Cross-based cost aggregation	3.99	3.39	9.94	5.20	8.89
Semiglobal matching	8.40	4.51	19.80	7.25	9.36
Interpolation	4.47	3.33	10.39	5.83	10.98
Subpixel Enhancement	4.02	3.28	9.44	5.03	8.91
Median filter	4.05	3.25	9.44	5.05	8.96
Bilateral filter	4.20	3.43	9.95	5.84	9.77
No stereo method	15.66	13.38	30.67	50.35	18.95
Full stereo method	3.99	3.25	9.44	5.03	8.89
	Middlebury				
	fst	acrt	sad	cens	ncc
Cross-based cost aggregation	9.87	10.63	43.09	29.28	33.89
Semiglobal matching	25.50	11.99	51.25	19.51	35.36
Interpolation	9.87	7.91	41.86	16.72	33.89
Subpixel Enhancement	10.29	8.44	42.71	17.18	34.12
Median filter	10.16	7.91	41.90	16.73	34.17
Bilateral filter	10.39	7.96	41.97	16.96	34.43
No stereo method	30.84	28.33	59.57	64.53	39.23
Full stereo method	9.87	7.91	41.86	16.72	33.89

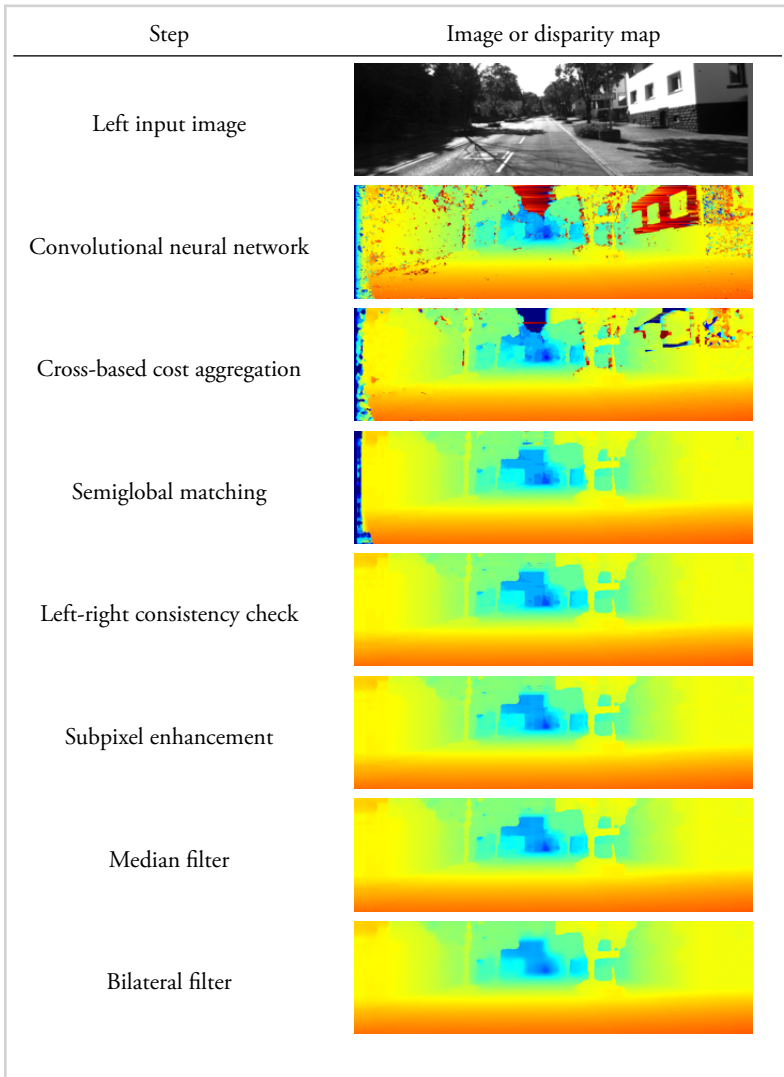


Figure 4.4

Comparison of disparity maps produced by different post-processing steps.

Table 4.10

The validation error as a function of training set size.

	KITTI 2012		KITTI 2015		Middlebury	
Data Set Size (%)	fst	acrt	fst	acrt	fst	acrt
20	3.17	2.84	4.13	3.53	11.14	9.73
40	3.11	2.75	4.10	3.40	10.35	8.71
60	3.09	2.67	4.05	3.34	10.14	8.36
80	3.05	2.65	4.02	3.29	10.09	8.21
100	3.02	2.61	3.99	3.25	9.87	7.91

2.61 % to 4.26 % on KITTI 2012, from 3.25 % to 4.51 % on KITTI 2015, and from 7.91 % to 11.99 % on Middlebury.

We did not use the left-right consistency check to eliminate errors in occluded regions on the Middlebury data set. The error rate increased from 7.91 % to 8.22 % using the left-right consistency check on the accurate architecture, which is why we decided to remove it.

4.9 Data Set Size

We used a supervised learning approach to measure the similarity between image patches. It is, therefore, natural to ask how does the size of the data set affect the quality of the disparity maps. To answer this question, we retrain our networks on smaller training sets obtained by selecting a random set of examples (see Table 4.10).

We observe that the validation error decreases as we increase the number of training examples. These experiments suggest a simple strategy for improving the results of our stereo method: collect a larger data set.

4.10 Transfer Learning

Up to this point the training and validation sets were created from the same stereo data set, either KITTI 2012, KITTI 2015, or Middlebury. To evaluate the performance of our method in the transfer learning setting, we run experiments where the validation error is computed on a different data set than the one used for training. For example,

Table 4.11

The validation error when the training and test sets differ. For example, the validation error is 3.16 % when the Middlebury data set is used for training the fast architecture and the trained network is tested on the KITTI 2012 data set.

		Test Set					
		KITTI 2012		KITTI 2015		Middlebury	
		fst	acrt	fst	acrt	fst	acrt
Training Set	KITTI 2012	3.02	2.61	4.12	3.99	12.78	11.09
	KITTI 2015	3.60	4.28	3.99	3.25	13.70	14.19
	Middlebury	3.16	3.07	4.48	4.49	9.87	7.91

we would use the Middlebury data set to train the matching cost neural network and evaluate its performance on the KITTI 2012 data set. These experiments give us some idea of the expected performance in a real-world application, where it isn't possible to train a specialized network because no ground truth is available. The results of these experiments are shown in Table 4.11.

Some results in Table 4.11 were unexpected. For example, the validation error on KITTI 2012 is lower when using the Middlebury training set compared to the KITTI 2015 training set, even though the KITTI 2012 data set is obviously more similar to KITTI 2015 than Middlebury. Furthermore, the validation error on KITTI 2012 is lower when using the fast architecture instead of the accurate architecture when training on KITTI 2015.

The matching cost neural network trained on the Middlebury data set transfers well to the KITTI data sets. Its validation error is similar to the validation errors obtained by networks trained on the KITTI data sets.

4.11 Hyperparameters

Searching for a good set of hyperparameters is a daunting task—with the search space growing exponentially with the number of hyperparameters and no gradient to guide us. To better understand the effect of each hyperparameter on the validation error, we conduct a series of experiments where we vary the value of a one hyperparameter while keeping the others fixed. The results are shown in Table 4.12 and can be summarized

Table 4.12

Validation errors computed across a range of hyperparameter settings.

Hyperparameter		KITTI 2012		KITTI 2015		Middlebury	
		fst	acrt	fst	acrt	fst	acrt
num_conv_layers	1	5.96	3.97	5.61	4.06	20.74	12.37
	2	3.52	2.98	4.19	3.45	12.11	9.20
	3	3.10	2.72	4.04	3.27	10.81	8.56
	4	3.02	2.61	3.99	3.25	10.26	8.21
	5	3.03	2.64	3.99	3.30	9.87	7.91
	6	3.05	2.70	4.01	3.38	9.71	8.11
num_conv_feature_maps	16	3.33	2.84	4.32	3.51	11.79	10.06
	32	3.15	2.68	4.12	3.35	10.48	8.67
	48	3.07	2.66	4.06	3.32	10.20	8.47
	64	3.02	2.64	3.99	3.30	9.87	8.12
	80	3.02	2.64	3.99	3.29	9.81	7.95
	96	2.99	2.68	3.97	3.27	9.62	8.03
	112	2.98	2.61	3.96	3.25	9.59	7.91
	128	2.97	2.63	3.95	3.23	9.45	7.92
num_fc_layers	1		2.83		3.50		8.52
	2		2.70		3.31		8.33
	3		2.62		3.30		8.06
	4		2.61		3.25		8.00
	5		2.62		3.29		7.91
num_fc_units	128		2.72		3.36		8.44
	256		2.65		3.28		8.03
	384		2.61		3.25		7.91
	512		2.60		3.23		7.90
dataset_neg_low	1.0	3.00	2.76	3.97	3.35	9.84	8.00
	1.5	3.00	2.71	3.97	3.33	9.87	7.91
	2.0	2.99	2.63	3.98	3.31	9.98	8.08
	4.0	3.02	2.61	3.99	3.25	10.20	8.66
	6.0	3.06	2.63	4.05	3.28	10.13	8.86
dataset_neg_high	6	3.00	2.72	3.98	3.30	9.87	8.59
	10	3.02	2.61	3.99	3.25	9.97	8.23
	14	3.04	2.61	4.02	3.25	10.00	8.05
	18	3.07	2.60	4.06	3.23	9.98	8.11
	22	3.07	2.61	4.05	3.24	10.16	7.91
dataset_pos	0.0	3.04	2.67	4.00	3.26	9.92	7.97
	0.5	3.02	2.65	3.99	3.28	9.87	7.91
	1.0	3.02	2.61	3.99	3.25	9.86	8.04
	1.5	3.04	2.62	4.04	3.27	10.00	8.34
	2.0	3.04	2.66	4.04	3.29	10.16	8.51

by observing that increasing the size of the network improves the generalization performance, but only up to a point, when presumably, because of the size of the data set, the generalization performance starts do decrease.

Note that the `num_conv_layers` hyperparameter implicitly controls the size of the image patches. For example, a network with one convolutional layer with 3×3 kernels compares image patches of size 3×3, while a network with five convolutional layers compares patches of size 11×11.

While this dissertation might give the impression that in order to get good results with neural networks it is necessary to search over a large space of architecture choices, possible loss functions, and hyperparameter values; and a large portion of my research time was, in fact, spent exploring this space. However, the following facts must not be overlooked:

- *The neural network for stereo is quite robust with respect to the values of hyperparameters.*

  Consider the accurate architecture on the KITTI 2012 dataset in Table 4.12. The validation error ranges from 2.61 % to 2.76 % for all tested values of the following hyperparameters: `num_fc_units`, `dataset_neg_high`, `dataset_neg_low` and `num_fc_units`. The difference between the worst and best performing networks in this case is marginal and only worth optimizing in a "competition setting".

  The parameters that can produce networks with high validation errors are `num_conv_layers`, `num_conv_feature_maps`, and `num_fc_layers`. We observed that the only networks with a high validation error were networks with less than three convolutional layers, less than three fully-connected layers, or networks with a small number of feature maps.

- *The search over the space of hyperparameters is not unique to neural networks.*

  The hyperparameter search problem would still be present if we used other supervised machine learning algorithms, for example logistic regression or support vector machines, instead of neural networks. In logistic regression we would have to choose the type and strength of regularization; in support vector machines the hyperparameters are the type and shape of the kernel and the strength of regularization. For example, Hsu et al. [70] recommended a brute-force

grid-search approach for fitting these two hyperparameters for support vector machines.

- *Most hyperparameters control the behaviour of the stereo method and not the neural network.*

  Consider all 21 hyperparameters listed in Table 4.6. The first six hyperparameters control the neural network, the next three are used for the dataset construction step, and the last twelve hyperparameters are used by the stereo method. From an engineering point of view, the stereo method was much harder to get working than the neural network training.

5

Conclusion

We presented two convolutional neural network architectures for learning a similarity measure on image patches and applied them to the problem of stereo matching.

The source code of our implementation is available online at https://github.com/jzbontar/mc-cnn. The online repository contains procedures for computing the disparity map, training the network, as well as the post-processing steps of the stereo method.

The accurate architecture produces disparity maps with lower error rates than any previously published method on the KITTI 2012, KITTI 2015, and Middlebury data sets. The fast architecture computes the disparity maps up to 90 times faster than the accurate architecture with only a small increase in error. These results suggest that convolutional neural networks are well suited for computing the stereo matching cost even for applications that require real-time performance.

The fact that a relatively simple convolutional neural network outperformed all previous methods on the well-studied problem of stereo is a rather important demonstration of the power of modern machine learning approaches.

5.1 Scientific Contributions

The main scientific contributions of this thesis are two new methods for computing the stereo matching cost. The contributions are itemized in the following list:

- *The accurate architecture—a convolutional neural network for computing the stereo matching cost optimized for accuracy.*

  The accurate architecture consists of two convolutional sub-network that extract feature vectors from small image patches. The feature vectors are forward-propagated through a multi-layer neural network that computes their similarity. The accurate architecture is trained on the KITTI and Middlebury stereo data sets to produce a good matching cost. Several post-processing steps are applied to improve the quality of the disparity maps. The error rates of the accurate architecture are 2.43 % on KITTI 2012, 3.89 % on KITTI 2015, and 8.29 % on the Middlebury data set. At the time of publication (November 2015), these were the lowest error rates on all three data sets.

  The proposed method was orally presented in the Reconstruction Meets Recognition Challenge workshop at the European Conference on Computer Vision

2014 in Zurich and was published in the Computer Vision and Pattern Recognition conference 2015 [154], which took place in Boston. An improved version of the accurate architecture is described in our Journal of Machine Learning Research paper in 2016 [155].

- *The fast architecture—a convolutional neural network for computing the stereo matching cost optimized for speed.*

  The accurate architecture produces precise disparity maps, however its range of application is limited to those that do not require real-time processing. After analyzing the network, we discovered that most of the time is spent in the multi-layer neural network with only a small fraction of time spent in the convolutional sub-networks. By replacing the multi-layer neural network with a fast operation—the dot product—we were able to compute the disparity maps up to 90 times faster. The error rates of the fast architecture are 2.82 % on KITTI 2012, 4.62 % on KITTI 2015, and 9.69 % on the Middlebury stereo data set. On small 320 × 240 resolution images with 32 disparity levels the output of the fast architecture is computed in 30 milliseconds, which is 60 times faster compared with the 1.8 seconds required by the accurate architecture.

  The fast architecture was first presented in our Journal of Machine Learning Research paper in 2016 [155].

- *The source code of both methods was made available under a permissive free software license.*

  The source code of both the fast and the accurate architecture is available online at `https://github.com/jzbontar/mc-cnn` under the permissive BSD licence. Several groups already use it in their work [1–3, 7, 54] and outperform our method on the KITTI and Middlebury data sets. At the time of writing (April 2016), the most accurate stereo methods on both the KITTI [54] and Middlebury [1] data sets use our code to compute the stereo matching cost. Releasing the source code was an important contribution to the field as it allowed other researchers to build on our work and push the boundaries of stereo matching.

The proposed stereo methods were thoroughly tested and compared with existing approaches. We evaluated different data augmentation steps, measured the runtime

over many hyperparameter settings, compared the error rate against established matching cost functions, evaluated the effect of each step of the post-processing, simulated experiments on smaller data sets, measured the network's performance on the transfer learning setting by training on one data set and testing on a different data set, and compared many different hyperparameter settings.

A

Disparity Maps

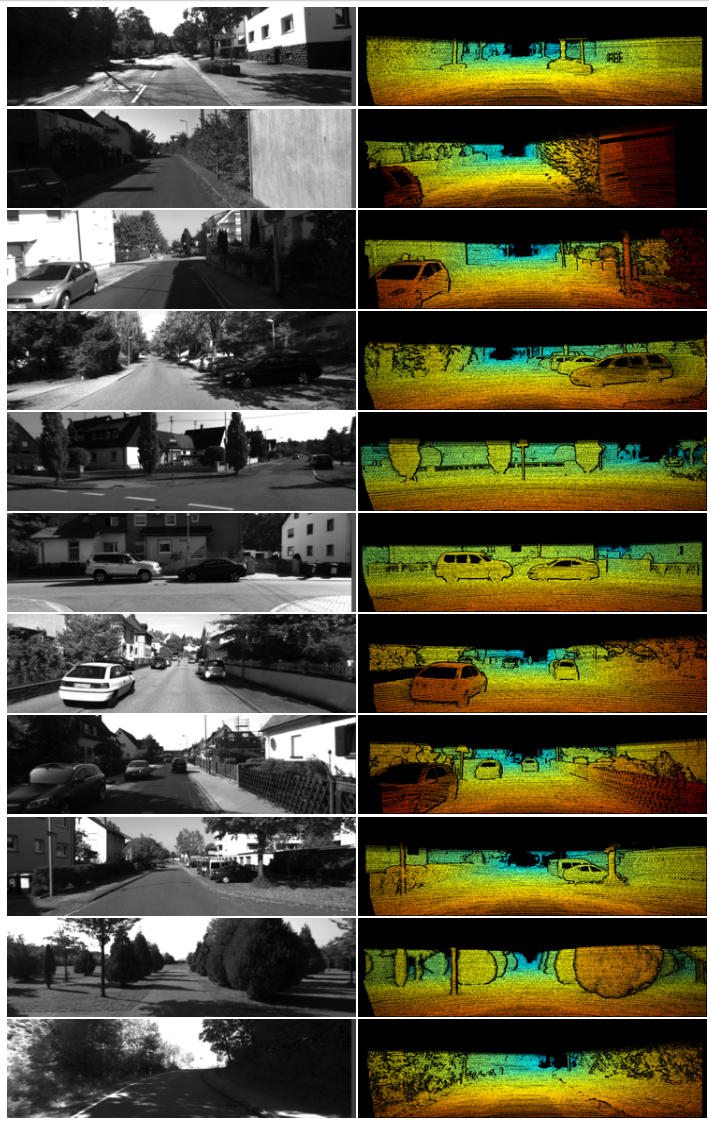


Figure A.1

Images and ground truth disparity maps from the KITTI 2012 data set.

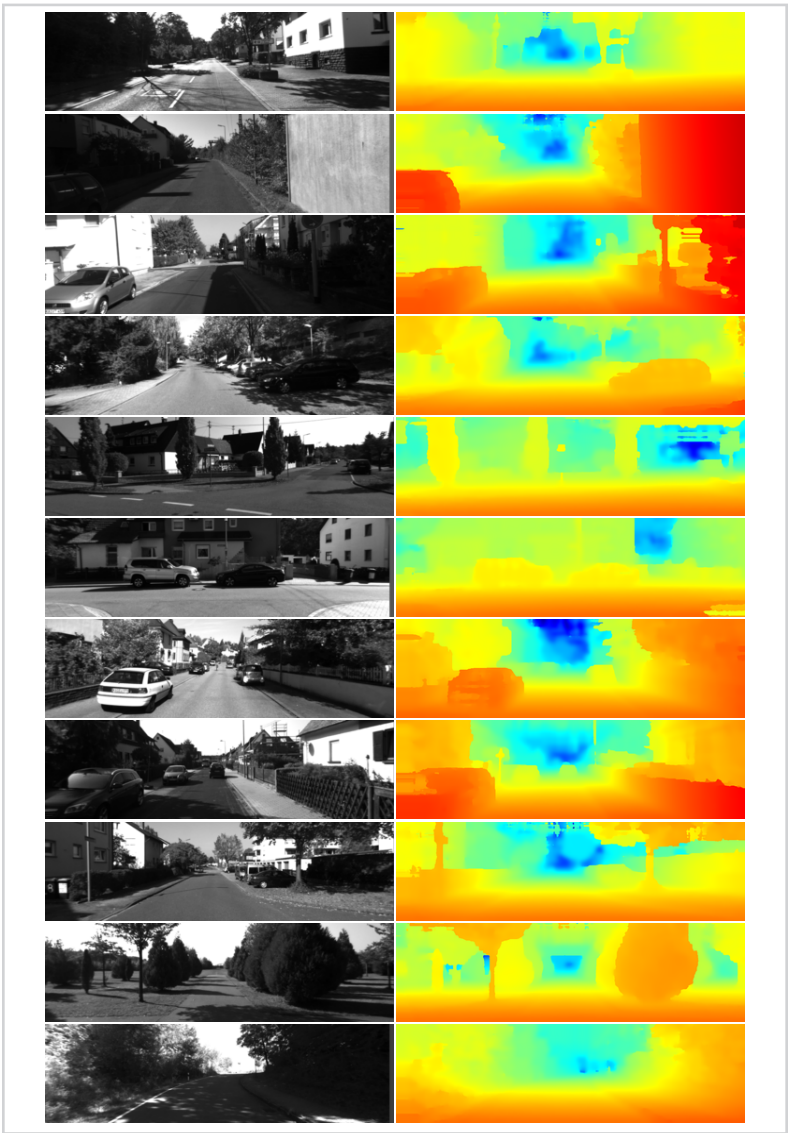


Figure A.2

Disparity maps produced by the fast architecture on examples from the KITTI 2012 data set.

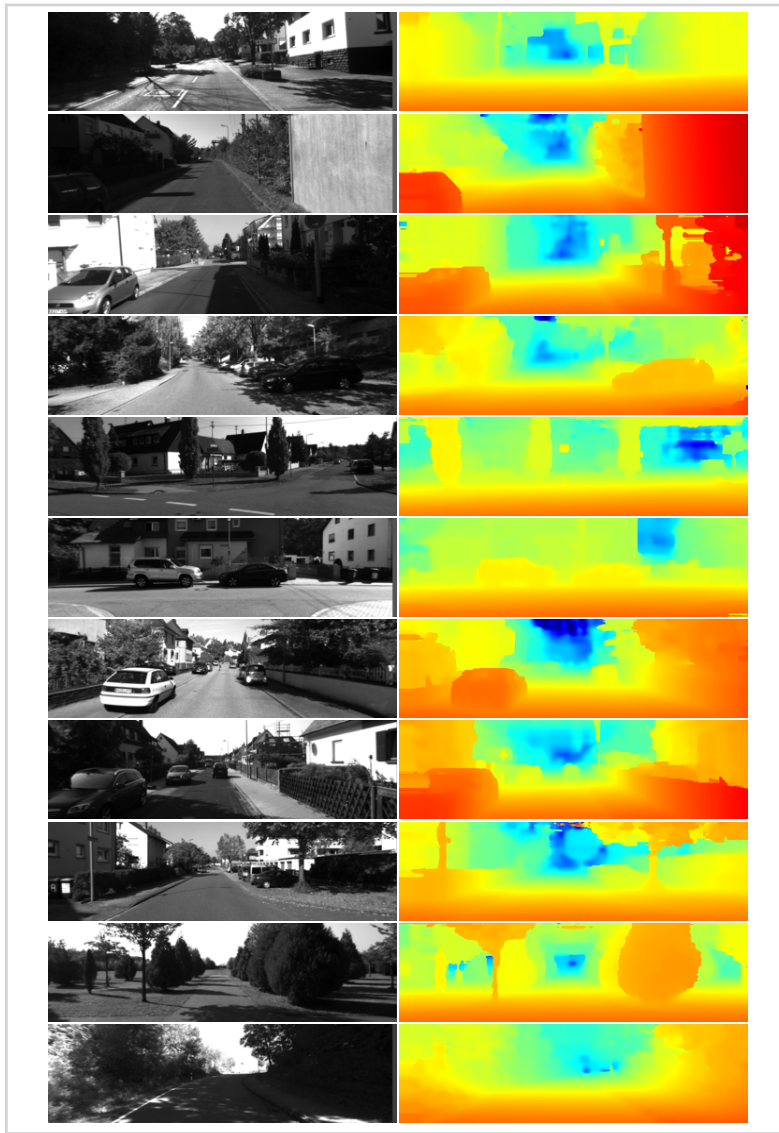


Figure A.3

Disparity maps produced by the accurate architecture on examples from the KITTI 2012 data set.

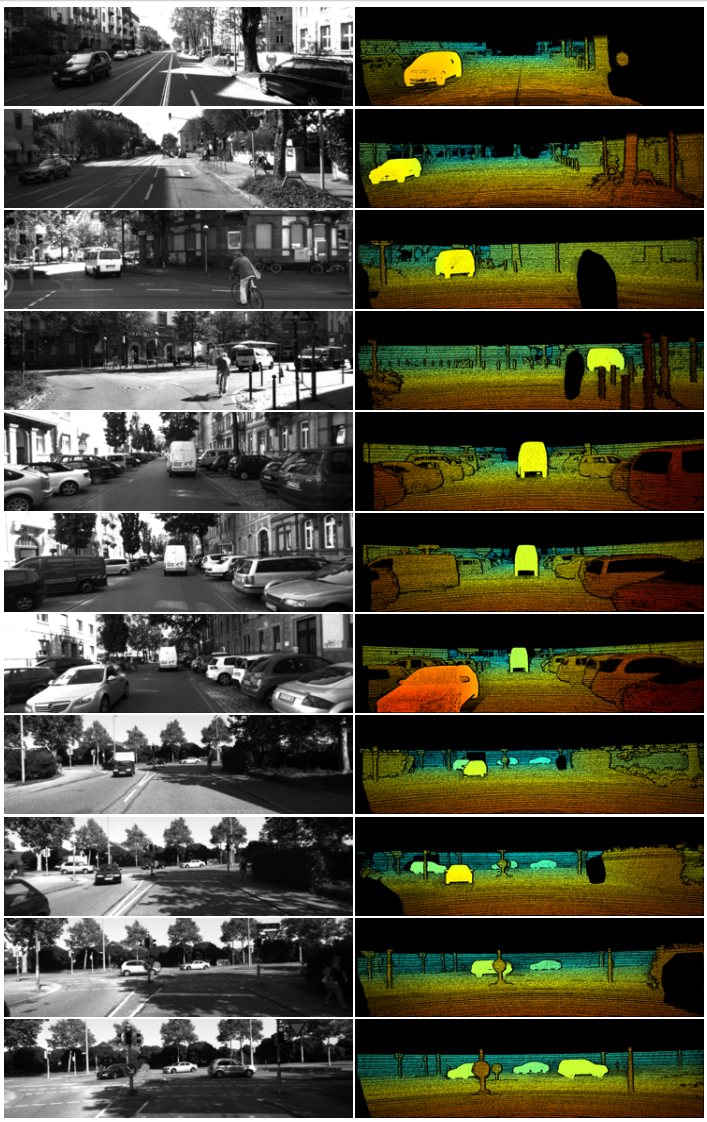


Figure A.4

Images and ground truth disparity maps from the KITTI 2015 data set.

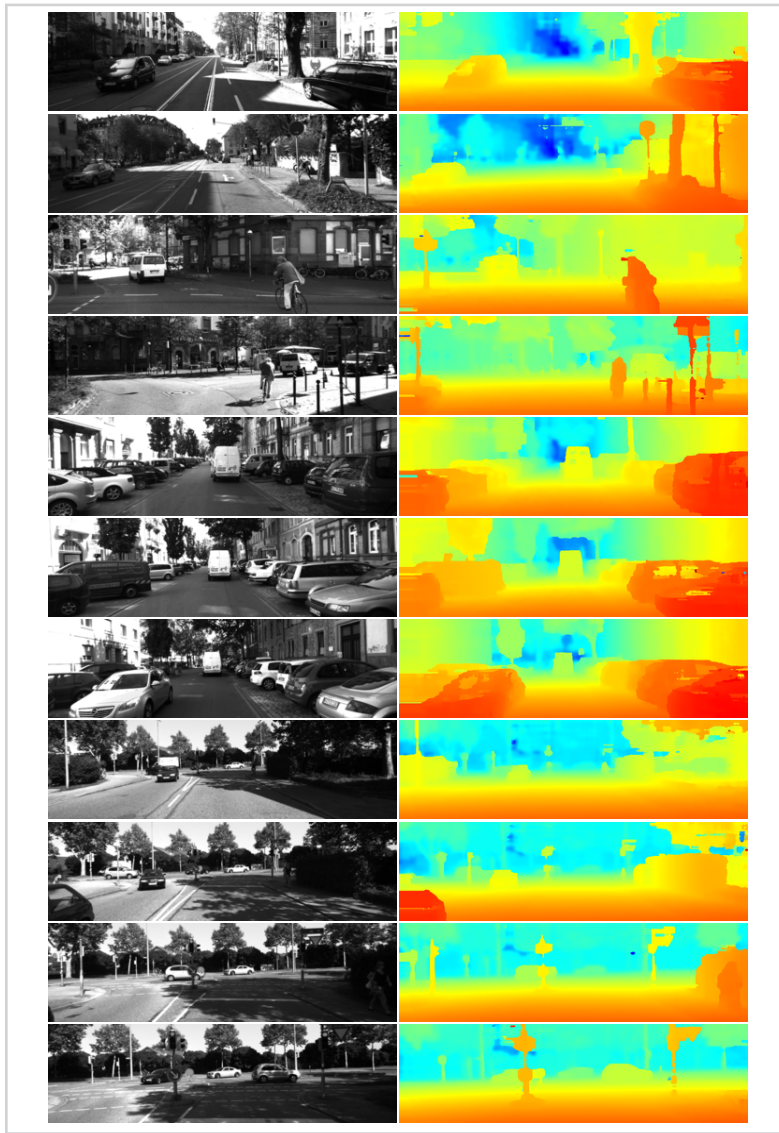


Figure A.5

Disparity maps produced by the fast architecture on examples from the KITTI 2015 data set.

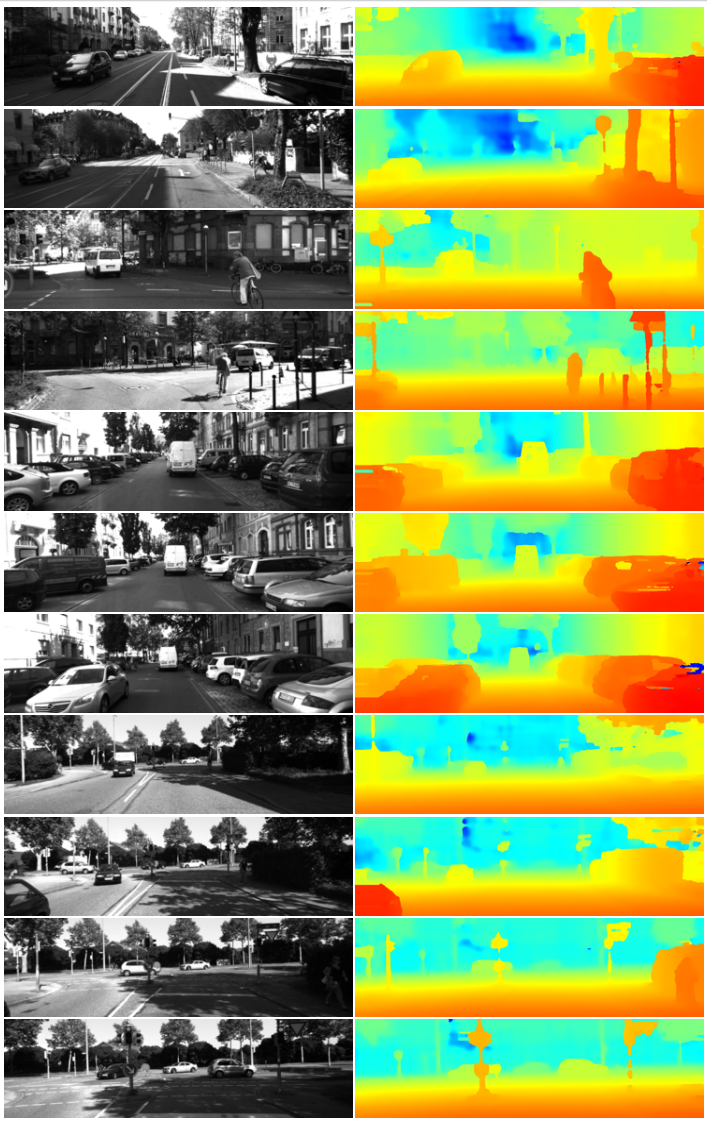


Figure A.6

Disparity maps produced by the accurate architecture on examples from the KITTI 2015 data set.

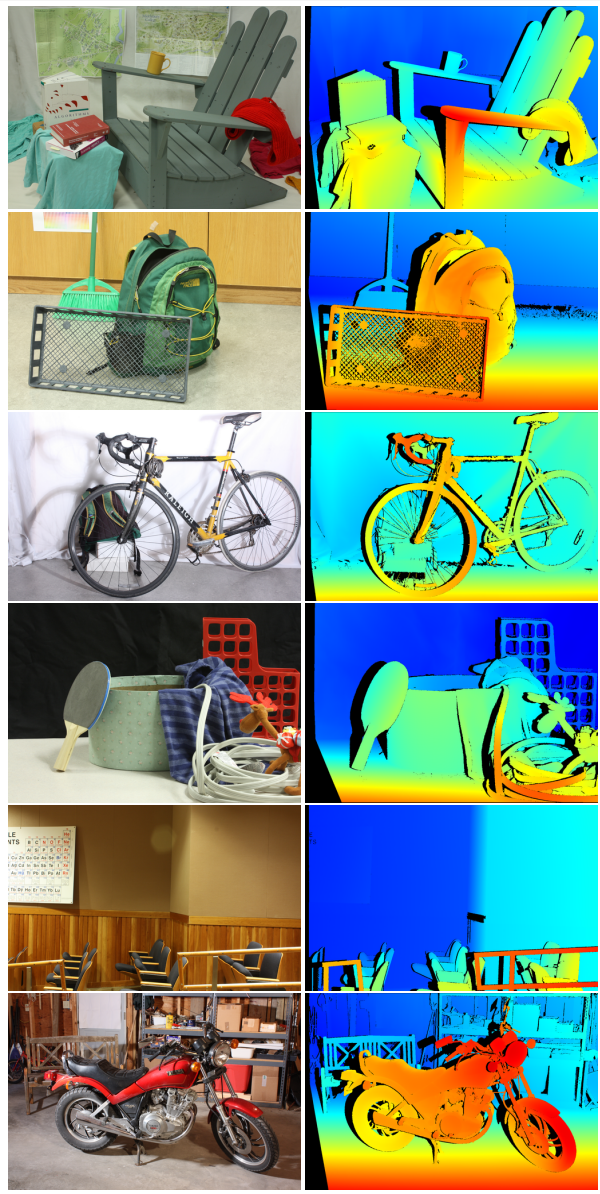


Figure A.7

Images and ground truth disparity maps from the Middlebury data set.

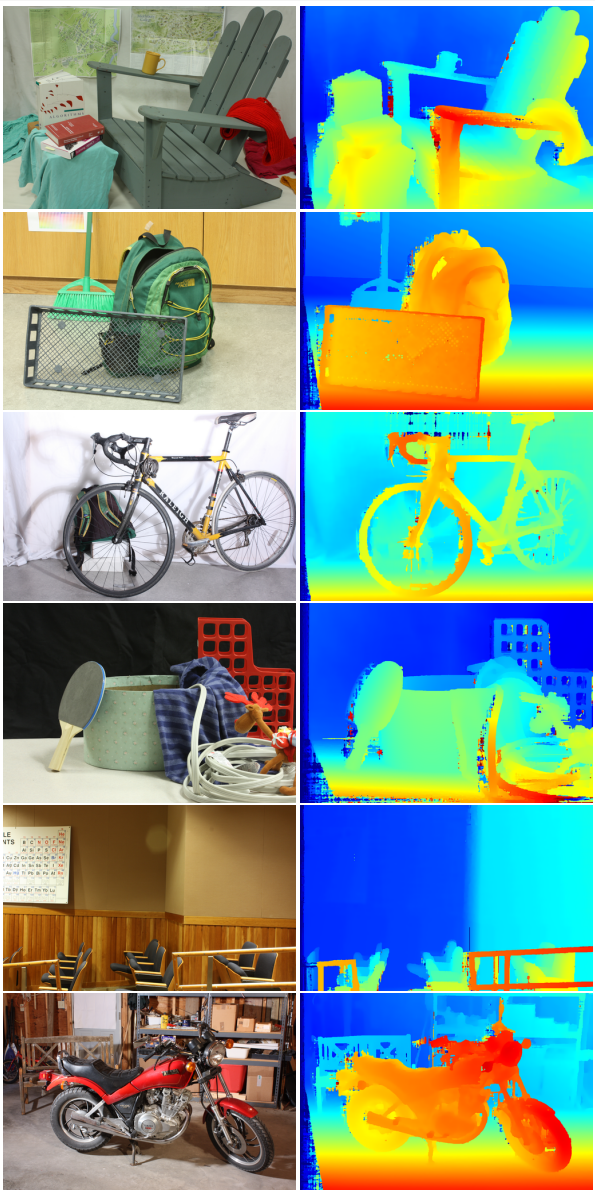


Figure A.8

Disparity maps produced by the fast architecture on examples from the Middlebury data set.

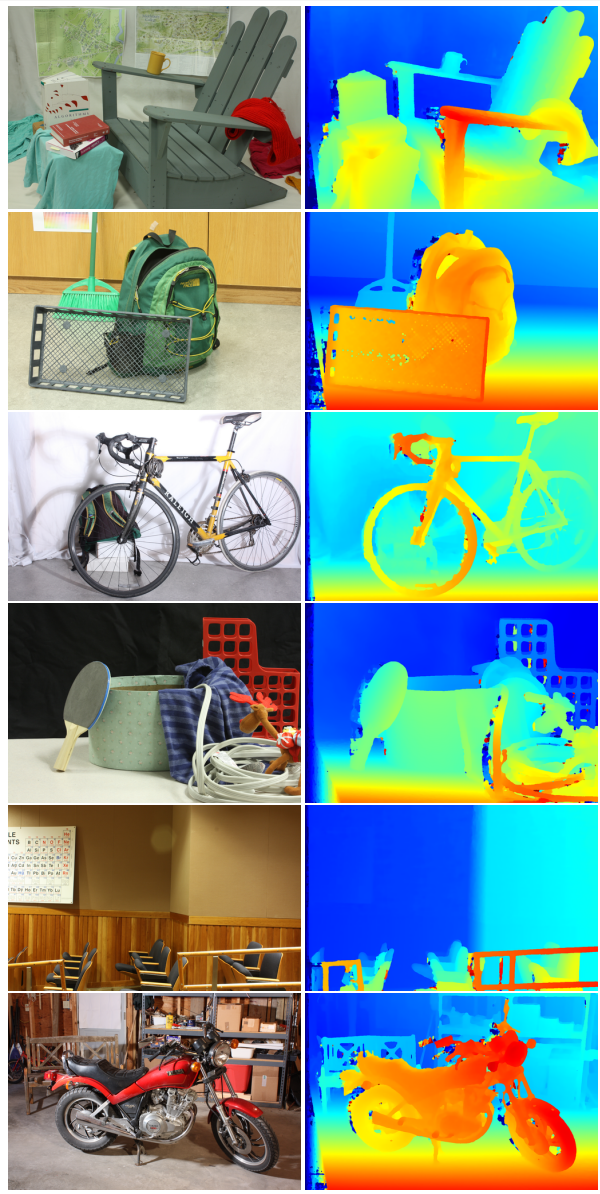


Figure A.9

Disparity maps produced by the accurate architecture on examples from the Middlebury data set.

Razširjeni povzetek

B

Z dvema kamerama, ki se razlikujeta le v vodoravni legi, ob istem času zajamemo dve sliki, levo in desno. Za vsak slikovni element leve slike želimo izračunati neskladnost, d. Neskladnost je razlika v vodoravni legi predmetov na levi in desni sliki – predmet, ki se na levi sliki nahaja na koordinatah (x, y), se na desni sliki nahaja na koordinatah $(x - d, y)$. Če poznamo neskladnost predmeta, lahko njegovo globino, z, izračunamo z uporabo sledeče enačbe:

$$z = \frac{fB}{d}, \tag{B.1}$$

kjer f označuje goriščno razdaljo kamer, B označuje razdaljo med kamerama in globina, z, predstavlja razdaljo predmeta od slikovne ravnine.

B.1 Nevronske Mreže

Nevronske mreže so družine modelov sestavljene iz več medsebojno povezanih procesnih enot ali nevronov [96, 108, 109, 146]. Nevron ima eno ali več vhodnih povezav in eno izhodno povezavo. Vhodne povezave so lahko izhodi drugih nevronov ali vhodni podatki, medtem ko so izhodne povezave lahko vhodi drugih nevronov ali predstavljajo izhod nevronske mreže. Vsak nevron izračuna uteženo vsoto števil na vhodu, vsoto preslika z nelinearno preslikavo in rezultat pošlje na izhod.

Različne arhitekture nevronski mrež delimo glede na to kako so nevroni med seboj povezani. V pričujočem delu se bomo osredotočili na večnivojske usmerjene nevronske mreže, v katerih so nevroni razdeljeni v več nivojev. Izhodi nevronov prvega nivoja so vhodi nevronov drugega nivoja, izhodi nevronov drugega nivoja so vhodi nevronov tretjega nivoja in tako dalje.

Večnivojska usmerjena nevronska mreža z N nivoji je funkcija, ki slika $\mathbb{R}^{D_0}$ v $\mathbb{R}^{D_N}$, kjer D_0 označuje dimenzijo vhodnega vektorja, D_N označuje dimenzijo izhodnega vektorja in D_k, za $k = 1, \ldots, N-1$, označuje število nevronov v k-tem nivoju. Funkcija večnivojske nevronske mreže je kompozitum funkcij na posameznih nivojih in je določena s sledečo enačbo:

$$y(\mathbf{x}) = (f^{(N)} \circ \ldots \circ f^{(1)})(\mathbf{x}), \tag{B.2}$$

kjer z $\mathbf{x}$ označimo vhodni vektor, z $f^{(k)}$ pa funkcijo k-tega nivoja. k-ti nivo nevronske mreže je funkcija, ki slika $\mathbb{R}^{D_{k-1}}$ v $\mathbb{R}^{D_k}$, in je definirana z enačbo

$$f_j^{(k)}(\mathbf{x}) = h\Big(\sum_{i=1}^{D_{k-1}} w_{ji}^{(k)} x_i + w_{j0}^{(k)}\Big), \tag{B.3}$$

kjer $f_j^{(k)}$ označuje j-ti element izračunanega vektorja, x_i označuje i-to komponento vhodnega vektorja, $w_{ji}^{(k)} \in \mathbb{R}$ označuje utež in h označuje izbrano nelinearno preslikavo. V našem delu smo uporabili $h_i(\mathbf{x}) = \max(0, x_i)$. Ker so nevroni nekega nivoja povezani z vsemi nevroni predhodnega nivoja pravimo takim nivojem polnopovezani.

Uteži nevronske mreže določimo z učenjem. Predpostavimo, da imamo na voljo učno množico – množico parov $\{(x^{(i)}, y^{(i)}); i = 1, \ldots, m\}$ – in funkcijo napake $E(\mathbf{w})$, ki napovedi nevronske mreže primerja z želenimi izhodi $y^{(i)}$. Učenje nevronske mreže je optimizacijski postopek katerega cilj je določiti vrednost uteži tako, da bo funkcija napake na učni množici čim manjša. Pogosto se za učenje uteži uporabi metoda gradientnega spusta. Na začetku vrednosti uteži nastavimo naključno, nato pa ponavljamo sledeči korak:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)}), \tag{B.4}$$

kjer $\mathbf{w}^{(\tau)}$ označuje vrednost uteži na koraku τ, $\eta > 0$ prilagaja hitrost učenja in $\nabla E(\mathbf{w}^{(\tau)})$ označuje gradient funkcije napake v točki $\mathbf{w}^{(\tau)}$. Gradient napake izračunamo z algoritmom vzvratnega razširjanja [109], ki z dosledno uporabo verižnega pravila za odvajanje in dinamičnega programiranja na učinkovit način izračuna gradient funkcije napake.

Konvolucijske nevronske mreže [88] so večnivojske nevronske mreže, pri katerih je vsaj en nivo konvolucijski. Konvolucijski nivo lahko uporabimo, ko je vhod podan v obliki strukturirane mreže. Primeri takšnega vhoda so zvočni zapis, slika in film. Konvolucijski nivo se od polnopovezanega razlikuje v dveh ključnih točkah: (1) nevroni so povezan le z nekaterimi nevroni predhodnega nivoja in (2) na vseh lokacija znotraj nivoja so uporabljene enake uteži.

B.2 Stereo Vid

Stereo vid je problem rekonstrukcije 3D modela prizora posnetega z dvema ali več kamerami.

B.2.1 Geometrijske Osnove

Analitični model kamere, ki ga v tem delu predpostavimo je kamera s točkasto odprtino. Transformacijo med točkami v prostoru in njihovo projekcijo opišemo z enačbo:

$$\mathbf{x} = \mathrm{P}\mathbf{X}, \tag{B.5}$$

kjer je $\mathbf{X} \in \mathbb{R}^4$ poljubna točka v prostoru, $\mathbf{x} \in \mathbb{R}^3$ je njena projekcija – obe točki sta predstavljeni v homogenih koordinatah – in $\mathtt{P} \in \mathbb{R}^{3\times4}$ je projekcijska matrika.

Predpostavimo, da z dvema kamerama opazujemo točko v prostoru $\mathbf{X}$. Naj se točka $\mathbf{X}$ preslika v točko $\mathbf{x}$ na prvi sliki in točko $\mathbf{x}'$ na drugi sliki. Velja torej $\mathbf{x} = \mathtt{P}\mathbf{X}$ in $\mathbf{x}' = \mathtt{P}'\mathbf{X}$, kjer je $\mathtt{P}$ projekcijska matrika prve kamere in $\mathtt{P}'$ projekcijska matrika druge kamere. Temeljna matrika $\mathtt{F} \in \mathbb{R}^{3\times3}$ opiše relacijo med točkama $\mathbf{x}$ in $\mathbf{x}'$ in sicer mora veljati:

$$\mathbf{x}'^T \mathtt{F} \mathbf{x} = 0. \tag{B.6}$$

Temeljno matriko lahko uporabimo tudi za izračun epipremice:

$$\mathbf{l}' = \mathtt{F}\mathbf{x}. \tag{B.7}$$

Za točko $\mathbf{x}'$ velja, da leži na epipremici $\mathbf{l}'$,

$$\mathbf{x}'^T \mathbf{l}' = 0. \tag{B.8}$$

Temeljno matriko izračunamo v postopku kalibracije kamer. Enačbi B.7 in B.8 sta za metode stereo vida izjemno pomembni, saj iskanje korespondenčne točke omejita na iskanje po epipremici.

B.2.2 Cena Ujemanja

Razvoj računskih metod za problem stereo vida se je začel okoli leta 1970. Barnard in Fischler [6] opišeta metode razvite v sedemdesetih letih, Dhond in Aggarwal [36] pa metode iz osemdesetih let. Dva novejša pregledna članka s področja sta izšla leta 2012 avtorjev Scharstein in Szeliski [113] in leta 2013 avtorja Brown s sodelavci [22]. Dober pregled področja je predstavljen v knjigi *Computer Vision: Algorithms and Applications* [131].

Funkcija za izračun cene ujemanja je funkcija, ki dve slikovni zaplati slika v realno število – ceno ujemanja. Cena ujemanja naj bo nizka natanko tedaj, ko je srednji element zaplat projekcija iste 3D točke.

V doktorski disertaciji opišemo in analiziramo sledeče metode za izračun cene ujemanja: metodo absolutnih razlik, vsoto absolutnih razlik, Birchfield-Tomasi [14], metodo mean filter, Laplacian of Gaussian, metodo BilSub avtorjev Ansar et al. [4], metodo ZSAD, metodo NCC in ZNCC, metodo Rank Filter avtorjev Zabih in Woodfil [152], metodo Soft Rank Filter, Census avtorjev Zabih and Woodfil [152], metodo Ordinal

measure avtorja Bhat et al. [13] in mero podobnosti, ki temelji na entropiji in sta jo predlagala Viola and Wells [139].

Natančnost naštetih metod ovrednotimo na stereo podatkovni množici Middlebury in ugotovimo, da je metoda Census najnatančnejša. Zato metodo, ki smo jo razvili tekom doktorske disertacije primerjamo tudi z metodo Census.

B.3 Konvolucijske Nevronske Mreže za Izračun Cene Ujemanja

Postopek za izračun globinske slike iz dveh kamer, ki temelji na uporabi konvolucijskih nevronskih mrež, je glavni doprinos pričujoče doktorske disertacije. Eden od pomembnejših korakov slehernega postopka za iskanje globinske slike je izračun cene ujemanja. V doktorski disertaciji pokažemo, da se lahko funkcijo za izračun cene ujemanja naučimo na množici označenih primerov z algoritmom nadzorovanega strojnega učenja.

Stereo podatkovne množice, na primer KITTI in Middlebury, sestojijo iz para slik in pravilne globinske slike. V podatkovni množici KITTI je pravilna globinska slika pridobljena z aktivnim senzorjem LIDAR, medtem ko je v podatkovni množici Middlebury izračunana z metodo strukturirane svetlobe.

Prvi korak predlagane metode je izgradnja učne množice za razvrščanje v dva razreda. Vsak učni primer je sestavljen iz para slikovnih zaplat, velikost 9×9 ali 11×11 slikovnih elementov. Ena slikovna zaplata je izrezana iz leve slike, druga iz desne slike. Razred učnega primera je določen glede na to ali sta slikovni zaplati v korespondenci. Na vsaki točki slike kjer poznamo pravilno neskladnost pridelamo dva učna primer. Prvi pripada pozitivnemu razredu, drugi pa negativnemu. Pozitivni primer dobimo tako, da slikovni zaplati izrežemo glede na pravilno neskladnost, negativnega pa tako, da pravilno neskladnost namenoma pokvarimo. Učna množica vsebuje 25 milijonov primerov, če za izgradnjo uporabimo podatkovno množico KITTI, oziroma 40 milijonov primerov, če uporabimo podatkovno množico Middlebury.

Za učenje preslikave uporabimo konvolucijsko nevronsko mrežo. V doktorski disertaciji opišemo dve arhitekturi mreže: prva izračuna globinsko sliko tudi do 90 krat hitreje od druge, vendar je globinska slika izračunana z drugo arhitekturo v povprečju bolj natančna.

Hitra arhitektura (*fast architecture*) temelji na siamski mreži [20]. Dve konvolucijski nevronski mreži, z enakimi utežmi, vhodni slikovni zaplati preslikata v vektorja značilk. Podobnost slikovnih zaplat izračunamo s kosinusom kota med vektorjema značilk.

Točna arhitektura nevronske mreže (*accurate architecture*) je podobna prvi. Slikovni zaplati z dvema konvolucijskima nevronskima mrežama predstavimo z vektorjem značilk. Vektorja značilk staknemo in posredujemo večnivojski usmerjeni nevronski mreži, ki je odgovorna za izračun podobnosti med vhodnima slikovnima zaplatama.

V disertaciji opišemo tudi učinkovit postopek za izračun globinske slike z opisanima arhitekturama. Izhod nevronske mreže obdelamo s številnimi koraki, ki popravijo globinsko sliko. Metode, ki smo jih uporabili za obdelavo izhoda nevronske mreže so sledeči: Cross-based Cost Aggregation avtorja Zhang et al. [157], Semiglobal Matching avtorja Hirschmüller [62], Left-Right Consistency Check in Subpixel Enhancement.

B.4 Rezultati

Natančnost pridelane globinske slike primerjamo z ostalimi, že uveljavljenimi, metodami na stereo podatkovnih množica KITTI 2012, KITTI 2015 in Middlebury.

Za množico testnih primerov nimamo pravilne globinske slike. Izhod naše metode ovrednotimo tako, da izračunane globinske slike naložimo na spletni strežnik (KITTI 2012[1], KITTI 2015[2] in Middlebury[3]), ki izračuna in vrne stopnjo natančnosti. Napaka globinske slike je razmerje slikovnih elementov kjer je absolutna razlika med napovedano neskladnostjo in pravilno neskladnostjo manjša ali enaka trem slikovnim elementom na podatkih KITTI, oziroma manjša ali enaka enemu slikovnemu elementu na podatkih Middlebury. Napaka hitre arhitekture je 2,43 % na KITTI 2012, 3,89 % na KITTI 2015 in 8,29 % na podatkovni množici Middlebury, medtem ko je napaka točne arhitekture 2,82 % na KITTI 2012, 4,62 % na KITTI 2015 in 9,69 % na podatkovni množici Middlebury. Ob času oddaje članka (oktober 2015) je bila naša metoda najnatančnejša na vseh treh podatkovnih množicah. Ob času pisanje doktorske disertacije (april 2016) pa so na prvih mestih metode, ki temeljijo na naši [1–3, 7, 54].

B.5 Zaključek

Predstavili smo dve arhitekturi za učenje mere podobnosti na slikovnih zaplatah, ki temeljita na uporabi globokih konvolucijskih nevronskih mrež. Naučeni meri podobnosti smo uporabili za izračun cene ujemanja pri problemu stereo vida.

[1] http://www.cvlibs.net/datasets/kitti/eval_stereo_flow.php?benchmark=stereo

[2] http://www.cvlibs.net/datasets/kitti/eval_scene_flow.php?benchmark=stereo

[3] http://vision.middlebury.edu/stereo/eval3/

Globinske slike izračunane z natančno arhitekturo dosežejo v povprečju manjšo napako od vseh objavljenih metod na stereo podatkovnih množicah KITTI 2012, KITTI 2015, in Middlebury. Siamska arhitektura je tudi do 90 krat hitrejša od natančne arhitekture, le z majhno izgubo točnosti globinskih slik. Predstavljeni rezultati dokazujejo, da so konvolucijske nevronske mreže primerne za izračun cene ujemanja in uporabne tudi v sistemih za delo v realnem času.

B.6 Prispevki k Znanosti

Najpomembnejša prispevka k znanosti sta metodi za izračun cene ujemanja pri problemu stereo vida. Prispevki so našteti v sledečem seznamu:

- *Točna arhitektura – konvolucijska nevronska mreža za izračun cene ujemanja s poudarkom na natančnosti izračunane globinske slike.*

  Točna arhitektura je sestavljena iz dveh konvolucijskih nevronskih mrež za izračun vektorja značilk in večnivojske usmerjene nevronske mreže za izračun podobnosti. Predstavljeno metodo smo ovrednotili na treh podatkovnih množicah: KITTI 2012, KITTI 2015 in Middlebury. Povprečne napake globinske slike so 2.43 % na podatkovni množici KITTI 2012, 3.89 % na podatkovni množici KITTI 2015 in 8.29 % na podatkovni množici Middlebury. Naša metoda je bila ob času objave (november 2015) najnatančnejša med vsemi objavljenimi metodami na vseh treh podatkovnih množica.

  Predlagana metoda je bila predstavljena na delavnici *Reconstruction Meets Recognition Challenge* konference *European Conference on Computer Vision* leta 2014 v Zürichu in objavljena na konferenci *Computer Vision and Pattern Recognition*, ki je potekala v Bostonu leta 2015 [154]. Izboljšana različica metode je opisana v članku, ki je bil sprejet v objavo v reviji *Journal of Machine Learning Research* [155].

- *Hitra arhitektura – konvolucijska nevronska mreža za izračun cene ujemanja s poudarkom na hitrosti izvajanja.*

  Globinska slika izračunana s točno arhitekturo je zelo natančna, ampak je, zaradi časa potrebnega za njen izračun, njihova uporaba omejena. Po temeljiti analizi izvajanja mreže smo ugotovili, da se večina časa porabi v večnivojski usmerjeni nevronski mreži. Ko smo večnivojsko usmerjeno nevronsko mrežo

zamenjali s hitro operacijo – s skalarnim produktom – se je čas izvajanja zmanjšal, tudi za 90 krat. Povprečne napake globinske slike so 2.82 % na podatkovni množici KITTI 2012, 4.62 % na podatkovni množici KITTI 2015 in 9.69 % na podatkovni množici Middlebury. Na slikah velikosti 320 × 240 slikovnih elementov z 32 nivoji neskladnosti hitra arhitektura za izračun potrebuje 30 milisekund, kar je 60 krat manj od natančne arhitekture, ki za izračun potrebuje 1.8 sekunde.

Predlagana metoda je predstavljena v članku, ki je bil sprejet v objavo v reviji *Journal of Machine Learning Research* [155].

- *Izvorna koda obeh metod je prosto dostopna in objavljena pod dovolilno licenco.*

  Objavili smo izvorno kodo obeh metod. Dostopna je na naslovu `https://github.com/jzbontar/mc-cnn` in je na voljo pod dovolilno licenco BSD. Izvorno kodo v svojem delu že sedaj uporabljajo druge raziskovalne skupine [1–3, 7, 54]. Ob času pisanja (april 2016) so na podatkovnih množica KITTI in Middlebury najnatančnejše metode, ki za izračun cene ujemanja uporabljajo našo izvorno kodo.

BIBLIOGRAPHY

[1] Anonymous. Adaptive smoothness constraints for efficient stereo matching using texture and edge information. *TBD*, 2016.

[2] Anonymous. Stereo depth map refinement with scene layout estimation. *TBD*, 2016.

[3] Anonymous. Very deep network for patch matching. *TBD*, 2016.

[4] Adnan Ansar, Andres Castano, and Larry Matthies. Enhanced real-time stereo using bilateral filtering. In *3D Data Processing, Visualization and Transmission, 2004. 3DPVT 2004. Proceedings. 2nd International Symposium on*, pages 455–462. IEEE, 2004.

[5] Stephen T Barnard. Stochastic stereo matching over scale. *International Journal of Computer Vision*, 3(1): 17–32, 1989.

[6] Stephen T Barnard and Martin A Fischler. Computational stereo. *ACM Computing Surveys (CSUR)*, 14 (4):553–572, 1982.

[7] Jonathan T Barron and Ben Poole. The fast bilateral solver. *arXiv preprint arXiv:1511.03296*, 2015.

[8] Yoshua Bengio and Yann LeCun. Scaling learning algorithms towards ai. *Large-Scale Kernel Machines*, 34, 2007.

[9] James R Bergen, Patrick Anandan, Keith J Hanna, and Rajesh Hingorani. Hierarchical model-based motion estimation. In *Computer Vision—ECCV'92*, pages 237–252. Springer, 1992.

[10] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13:281–305, 2012.

[11] James Bergstra, Rémi Bardenet, Yoshua Bengio, Balázs Kégl, et al. Algorithms for hyper-parameter optimization. In *25th Annual Conference on Neural Information Processing Systems (NIPS 2011)*, 2011.

[12] Julian Besag. On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 259–302, 1986.

[13] Dinkar N Bhat, Shree K Nayar, and Alok Gupta. Motion estimation using ordinal measures. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pages 982–987. IEEE, 1997.

[14] Stan Birchfield and Carlo Tomasi. A pixel dissimilarity measure that is insensitive to image sampling. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(4):401–406, 1998.

[15] Chris M Bishop. *Pattern recognition and machine learning*, volume 4. springer New York, 2006.

[16] Robert C Bolles, Harlyn H Baker, and Marsha Jo Hannah. The jisct stereo evaluation. In *DARPA Image Understanding Workshop*, pages 263–274, 1993.

[17] Yuri Boykov, Olga Veksler, and Ramin Zabih. Markov random fields with efficient approximations. In *Computer vision and pattern recognition, 1998. Proceedings. 1998 IEEE computer society conference on*, pages 648–655. IEEE, 1998.

[18] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 23(11):1222–1239, 2001.

[19] G. Bradski. The OpenCV library. *Dr. Dobb's Journal of Software Tools*, 2000.

[20] Jane Bromley, James W Bentz, Léon Bottou, Isabelle Guyon, Yann LeCun, Cliff Moore, Eduard Säckinger, and Roopak Shah. Signature verification using a siamese time delay neural network. *International Journal of Pattern Recognition and Artificial Intelligence*, 7 (04):669–688, 1993.

[21] Matthew Brown, Gang Hua, and Simon Winder. Discriminative learning of local image descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):43–57, 2011.

[22] Myron Z Brown, Darius Burschka, and Gregory D Hager. Advances in computational stereo. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(8):993–1008, 2003.

[23] Thomas Brox and Jitendra Malik. Large displacement optical flow: descriptor matching in variational motion estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(3):500–513, 2011.

[24] Jan Čech and Radim Šára. Complex correlation statistic for dense stereoscopic matching. In *Image Analysis*, pages 598–608. Springer, 2005.

[25] Ayan Chakrabarti, Ying Xiong, Steven J. Gortler, and Todd Zickler. Low-level vision by consensus in a spatial hierarchy of regions. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[26] Zhuoyuan Chen, Xun Sun, Yinan Yu, Liang Wang, and Chang Huang. A deep visual correspondence embedding model for stereo matching costs. *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[27] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cuDNN: Efficient primitives for deep learning. *CoRR*, abs/1410.0759, 2014. URL http://arxiv.org/abs/1410.0759.

[28] Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 539–546. IEEE, 2005.

[29] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surface of multilayer networks. *arXiv preprint arXiv:1412.0233*, 2014.

[30] Dan Ciresan, Alessandro Giusti, Luca M Gambardella, and Jürgen Schmidhuber. Deep neural networks segment neuronal membranes in electron microscopy images. In *Advances in neural information processing systems*, pages 2843–2851, 2012.

[31] Dan Cireşan, Ueli Meier, Jonathan Masci, and Jürgen Schmidhuber. Multi-column deep neural network for traffic sign classification. *Neural Networks*, 32: 333–338, 2012.

[32] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.

[33] Camille Couprie, Clément Farabet, Laurent Najman, and Yann LeCun. Indoor semantic segmentation using depth information. *arXiv preprint arXiv:1301.3572*, 2013.

[34] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[35] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7:1–30, 2006.

[36] Umesh R Dhond and Jake K Aggarwal. Structure from stereo-a review. *IEEE transactions on systems, man, and cybernetics*, 19(6):1489–1510, 1989.

[37] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2650–2658, 2015.

[38] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. In *Advances in Neural Information Processing Systems*, pages 2366–2374, 2014.

[39] Nils Einecke and Julian Eggert. A two-stage correlation method for stereoscopic depth estimation. In *Digital Image Computing: International Conference on Techniques and Applications (DICTA)*, pages 227–234, 2010.

[40] Clement Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. Learning hierarchical features for scene labeling. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(8):1915–1929, 2013.

[41] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient belief propagation for early vision. *International journal of computer vision*, 70(1):41–54, 2006.

[42] Philipp Fischer, Alexey Dosovitskiy, Eddy Ilg, Philip Häusser, Caner Hazırbaş, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. *arXiv preprint arXiv:1504.06852*, 2015.

[43] William T Freeman, Egon C Pasztor, and Owen T Carmichael. Learning low-level vision. *International journal of computer vision*, 40(1):25–47, 2000.

[44] Kunihiko Fukushima. Neural network model for a mechanism of pattern recognition unaffected by shift in position—neocognitron. *Electron. & Commun. Japan*, 62(10):11–18, 1979.

[45] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[46] Kunihiko Fukushima. Increasing robustness against background noise: Visual pattern recognition by a neocognitron. *Neural networks*, 24(7):767–778, 2011.

[47] Kunihiko Fukushima. Artificial vision by multi-layered neural networks: Neocognitron and its advances. *Neural Networks*, 37:103–119, 2013.

[48] Andreas Geiger, Martin Roser, and Raquel Urtasun. Efficient large-scale stereo matching. In *Proceedings of the 10th Asian Conference on Computer Vision - Volume Part I*, ACCV'10, pages 25–38. Springer-Verlag, Berlin, Heidelberg, 2011.

[49] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: the KITTI dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[50] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, pages 721–741, 1984.

[51] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*, 2010.

[52] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016. URL http://goodfeli.github.io/dlbook/.

[53] DM Greig, BT Porteous, and Allan H Seheult. Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 271–279, 1989.

[54] Fatma Güney and Andreas Geiger. Displets: Resolving stereo ambiguities using object knowledge. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[55] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *Computer vision and pattern recognition, 2006 IEEE computer society conference on*, volume 2, pages 1735–1742. IEEE, 2006.

[56] Raia Hadsell, Pierre Sermanet, Jan Ben, Ayse Erkan, Marco Scoffier, Koray Kavukcuoglu, Urs Muller, and Yann LeCun. Learning long-range vision for autonomous off-road driving. *Journal of Field Robotics*, 26(2):120–144, 2009.

[57] Ralf Haeusler, Rahul Nair, and Daniel Kondermann. Ensemble learning for confidence measures in stereo vision. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2013.

[58] Xufeng Han, Thomas Leung, Yangqing Jia, Rahul Sukthankar, and Alexander C Berg. MatchNet: Unifying feature and metric learning for patch-based matching. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[59] Marsha J Hannah. Computer matching of areas in stereo images. Technical report, DTIC Document, 1974.

[60] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[61] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[62] Heiko Hirschmüller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2): 328–341, 2008.

[63] Heiko Hirschmüller and Daniel Scharstein. Evaluation of cost functions for stereo matching. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007.

[64] Heiko Hirschmüller and Daniel Scharstein. Evaluation of stereo matching costs on images with radiometric differences. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(9):1582–1599, 2009.

[65] Heiko Hirschmüller, Peter R Innocent, and Jon Garibaldi. Real-time correlation-based stereo vision with reduced border errors. *International Journal of Computer Vision*, 47(1-3):229–246, 2002.

[66] Dorit S Hochbaum. An efficient algorithm for image segmentation, markov random fields and related problems. *Journal of the ACM (JACM)*, 48(4): 686–701, 2001.

[67] Michael Hornacek, Andrew Fitzgibbon, and Carsten Rother. SphereFlow: 6 DoF scene flow from RGB-D pairs. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[68] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2): 251–257, 1991.

[69] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[70] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. A practical guide to support vector classification. 2003.

[71] David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106–154, 1962.

[72] David H Hubel and Torsten N Wiesel. Receptive fields and functional architecture in two nonstriate visual areas (18 and 19) of the cat. *Journal of neurophysiology*, 28(2):229–289, 1965.

[73] Hiroshi Ishikawa. Exact optimization for markov random fields with convex priors. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(10): 1333–1336, 2003.

[74] Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2146–2153. IEEE, 2009.

[75] Junhwan Kim, Vladimir Kolmogorov, and Ramin Zabih. Visual correspondence using energy minimization and mutual information. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 1033–1040. IEEE, 2003.

[76] Scott Kirkpatrick. Optimization by simulated annealing: Quantitative studies. *Journal of statistical physics*, 34(5-6):975–986, 1984.

[77] Vladimir Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(10):1568–1583, 2006.

[78] Vladimir Kolmogorov and Ramin Zabin. What energy functions can be minimized via graph cuts? *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(2):147–159, 2004.

[79] Dan Kong and Hai Tao. A method for learning matching errors for stereo computation. *British Machine Vision Conference (BMVC)*, 2004.

[80] Dan Kong and Hai Tao. Stereo matching via learning multiple experts behaviors. *British Machine Vision Conference (BMVC)*, 2006.

[81] Kurt Konolige. Small vision systems: Hardware and implementation. In *Robotics Research*, pages 203–212. Springer, 1998.

[82] Jana Kostková and Radim Sára. Stratified dense matching for stereopsis in complex scenes. *British Machine Vision Conference (BMVC)*, 2003.

[83] Jedrzej Kowalczuk, Eric T Psota, and Lance C Perez. Real-time stereo matching on CUDA using an iterative refinement method for adaptive support-weight correspondences. *IEEE Transactions on Circuits and Systems for Video Technology*, 23(1):94–104, 2013.

[84] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1106–1114, 2012.

[85] Steve Lawrence, C Lee Giles, Ah Chung Tsoi, and Andrew D Back. Face recognition: A convolutional neural-network approach. *Neural Networks, IEEE Transactions on*, 8(1):98–113, 1997.

[86] Yann LeCun. Une procedure d'apprentissage pour reseau a seuil asymmetrique (a learning scheme for asymmetric threshold networks). 1985.

[87] Yann LeCun. A theoretical framework for backpropagation. In *The Connectionist Models Summer School*, volume 1, pages 21–28, 1988.

[88] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86 (11):2278–2324, 1998.

[89] Yann LeCun, Léon Bottou, Genevieve Orr, and Klaus Müller. Efficient backprop. *Neural networks: Tricks of the trade*, pages 546–546, 1998.

[90] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[91] Yunpeng Li and Daniel P Huttenlocher. Learning for stereo vision using the structured support vector machine. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2008.

[92] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *arXiv preprint arXiv:1411.4038*, 2014.

[93] David Marr and Tomaso Poggio. Cooperative computation of stereo disparity. *Science*, 194(4262): 283–287, 1976.

[94] Jose Marroquin, Sanjoy Mitter, and Tomaso Poggio. Probabilistic solution of ill-posed problems in computational vision. *Journal of the american statistical association*, 82(397):76–89, 1987.

[95] Nikolaus Mayer, Eddy Ilg, Philip Häusser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. *arXiv preprint arXiv:1512.02134*, 2015.

[96] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[97] Xing Mei, Xun Sun, Mingcai Zhou, Haitao Wang, Xiaopeng Zhang, et al. On building an accurate stereo matching system on graphics hardware. *IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 467–474, 2011.

[98] Moritz Menze and Andreas Geiger. Object scene flow for autonomous vehicles. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[99] Hans P. Morevec. Towards automatic visual obstacle avoidance. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence - Volume 2*, pages 584–584. Morgan Kaufmann Publishers Inc., 1977.

[100] Urs Muller, Jan Ben, Eric Cosatto, Beat Flepp, and Yann L Cun. Off-road obstacle avoidance through end-to-end learning. In *Advances in neural information processing systems*, pages 739–746, 2005.

[101] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with CUDA. *Queue*, 6(2):40–53, 2008.

[102] Mattis Paulin, Matthijs Douze, Zaid Harchaoui, Julien Mairal, Florent Perronin, and Cordelia Schmid. Local convolutional features with unsupervised training for image retrieval. In *IEEE International Conference on Computer Vision (ICCV)*, pages 91–99, 2015.

[103] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 2014.

[104] Martin Peris, Atsuto Maki, Sara Martull, Yasuhiro Ohkawa, and Kazuhiro Fukui. Towards a simulation driven stereo vision system. In *21st International Conference on Pattern Recognition (ICPR)*, pages 1038–1042, 2012.

[105] Eric T Psota, Jedrzej Kowalczuk, Mateusz Mittek, and Lance C Perez. Map disparity estimation using hidden markov trees. *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[106] Lynn H Quam and Artificial Intelligence Center. Hierarchical warp stereo. *Readings in computer vision*, pages 80–86, 1984.

[107] Jerome Revaud, Philippe Weinzaepfel, Zaid Harchaoui, and Cordelia Schmid. Deepmatching: Hierarchical deformable dense matching. *ArXiv e-prints*, 1(7):8, 2015.

[108] Frank Rosenblatt. Principles of neurodynamics. 1962.

[109] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

[110] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

[111] Daniel Scharstein and Chris Pal. Learning conditional random fields for stereo. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2007.

[112] Daniel Scharstein and Richard Szeliski. Stereo matching with nonlinear diffusion. *International journal of computer vision*, 28(2):155–174, 1998.

[113] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47(1-3):7–42, 2002.

[114] Daniel Scharstein and Richard Szeliski. High-accuracy stereo depth maps using structured light. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2003.

[115] Daniel Scharstein, Heiko Hirschmüller, York Kitajima, Greg Krathwohl, Nera Nešić, Xi Wang, and Porter Westling. High-resolution stereo datasets with subpixel-accurate ground truth. *German Conference on Pattern Recognition (GCPR)*, September 2014.

[116] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[117] Pierre Sermanet and Yann LeCun. Traffic sign recognition with multi-scale convolutional networks. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 2809–2813. IEEE, 2011.

[118] Pierre Sermanet, Raia Hadsell, Marco Scoffier, Matt Grimes, Jan Ben, Ayse Erkan, Chris Crudele, Urs Miller, and Yann LeCun. A multirange architecture for collision-free off-road robot navigation. *Journal of Field Robotics*, 26(1):52–87, 2009.

[119] Pierre Sermanet, Soumith Chintala, and Yann LeCun. Convolutional neural networks applied to house numbers digit classification. In *Pattern Recognition (ICPR), 2012 21st International Conference on*, pages 3288–3291. IEEE, 2012.

[120] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[121] Naum Zuselevich Shor. *Minimization methods for non-differentiable functions*. Springer-Verlang, 1985.

[122] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[123] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Learning local feature descriptors using convex optimisation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(8):1573–1585, 2014.

[124] Sudipta N Sinha, Daniel Scharstein, and Richard Szeliski. Efficient high-resolution stereo matching using local plane sweeps. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[125] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *arXiv preprint arXiv:1206.2944*, 2012.

[126] Aristotle Spyropoulos, Nikos Komodakis, and Philippos Mordohai. Learning to detect ground control points for improving the accuracy of stereo matching. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[127] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15 (1):1929–1958, 2014.

[128] Harold S Stone. Multiprocessor scheduling with the aid of network flow algorithms. *Software Engineering, IEEE Transactions on*, pages 85–93, 1977.

[129] Deqing Sun, Stefan Roth, and Michael J Black. A quantitative analysis of current practices in optical flow estimation and the principles behind them. *International Journal of Computer Vision*, 106(2): 115–137, 2014.

[130] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014.

[131] Richard Szeliski. *Computer vision: algorithms and applications*. Springer, 2010.

[132] Richard Szeliski, Ramin Zabih, Daniel Scharstein, Olga Veksler, Vladimir Kolmogorov, Aseem Agarwala, Marshall Tappen, and Carsten Rother. A comparative study of energy minimization methods for markov random fields. In *Computer Vision–ECCV 2006*, pages 16–29. Springer, 2006.

[133] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lars Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 1701–1708. IEEE, 2014.

[134] Carlo Tomasi and Roberto Manduchi. Bilateral filtering for gray and color images. In *Computer Vision, 1998. Sixth International Conference on*, pages 839–846. IEEE, 1998.

[135] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christopher Bregler. Efficient object localization using convolutional networks. *arXiv preprint arXiv:1411.4280*, 2014.

[136] Jonathan Tompson, Arjun Jain, Yann LeCun, and Christoph Bregler. Joint training of a convolutional network and a graphical model for human pose estimation. *arXiv preprint arXiv:1406.2984*, 2014.

[137] Tomasz Trzcinski, Mario Christoudias, Vincent Lepetit, and Pascal Fua. Learning image descriptors with the boosting-trick. In *Advances in neural information processing systems*, pages 269–277, 2012.

[138] Srinivas C Turaga, Joseph F Murray, Viren Jain, Fabian Roth, Moritz Helmstaedter, Kevin Briggman, Winfried Denk, and H Sebastian Seung. Convolutional networks can learn to generate affinity graphs for image segmentation. *Neural Computation*, 22(2): 511–538, 2010.

[139] Paul Viola and William M Wells. Alignment by maximization of mutual information. *International journal of computer vision*, 24(2):137–154, 1997.

[140] Christoph Vogel, Konrad Schindler, and Stefan Roth. Piecewise rigid scene flow. *IEEE International Conference on Computer Vision (ICCV)*, 2013.

[141] Christoph Vogel, Stefan Roth, and Konrad Schindler. View-consistent 3D scene flow estimation over multiple frames. *European Conference on Computer Vision (ECCV)*, September 2014.

[142] Christoph Vogel, Konrad Schindler, and Stefan Roth. 3D scene flow estimation with a piecewise rigid scene model. *International Journal of Computer Vision*, pages 1–28, 2015.

[143] Yair Weiss and William T Freeman. Correctness of belief propagation in gaussian graphical models of arbitrary topology. *Neural computation*, 13(10): 2173–2200, 2001.

[144] Paul J Werbos. Applications of advances in nonlinear sensitivity analysis. In *System modeling and optimization*, pages 762–770. Springer, 1982.

[145] Jason Weston, Samy Bengio, and Nicolas Usunier. Wsabie: Scaling up to large vocabulary image annotation. In *IJCAI*, volume 11, pages 2764–2770, 2011.

[146] Bernard Widrow and Marcian E Hoff. Adaptive switching circuits. 1960.

[147] Andrew Witkin, Demetri Terzopoulos, and Michael Kass. Signal matching through scale space. *International Journal of Computer Vision*, 1(2):133–144, 1987.

[148] Koichiro Yamaguchi, David McAllester, and Raquel Urtasun. Efficient joint segmentation, occlusion labeling, stereo and flow estimation. *European Conference on Computer Vision (ECCV)*, September 2014.

[149] Daniel Yamins, David Tax, and James S Bergstra. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 115–123, 2013.

[150] Jonathan S Yedidia, William T Freeman, Yair Weiss, et al. Generalized belief propagation. In *NIPS*, volume 13, pages 689–695, 2000.

[151] Dong Yu, Li Deng, and Frank Seide. The deep tensor neural network with applications to large vocabulary speech recognition. *IEEE Trans. on Audio, Speech, and Language Processing*, 2(3):388–396, 2013.

[152] Ramin Zabih and John Woodfill. Non-parametric local transforms for computing visual correspondence. *European Conference on Computer Vision (ECCV)*, 1994.

[153] Sergey Zagoruyko and Nikos Komodakis. Learning to compare image patches via convolutional neural networks. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[154] Jure Žbontar and Yann LeCun. Computing the stereo matching cost with a convolutional neural network. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[155] Jure Žbontar and Yann LeCun. Stereo matching by training a convolutional neural network to compare image patches. *arXiv preprint arXiv:1510.05970*, 2015.

[156] Chi Zhang, Zhiwei Li, Yanhua Cheng, Rui Cai, Hongyang Chao, and Yong Rui. Meshstereo: A global stereo model with mesh alignment regularization for view interpolation. *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[157] Ke Zhang, Jiangbo Lu, and Gauthier Lafruit. Cross-based local stereo matching using orthogonal integral images. *IEEE Transactions on Circuits and Systems for Video Technology*, 19(7):1073–1079, 2009.

[158] Li Zhang and Steven M Seitz. Estimating optimal parameters for MRF stereo from a single image pair. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(2):331–342, 2007.

[159] Yan Zhu, Yuandong Tian, Dimitris Mexatas, and Piotr Dollár. Semantic amodal segmentation. *arXiv preprint arXiv:1509.01329*, 2015.

[160] C Lawrence Zitnick and Takeo Kanade. A cooperative algorithm for stereo matching and occlusion detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(7):675–684, 2000.